

AutoCAD® 2002

autodesk®

**ActiveX and VBA
Developers' Guide**

Copyright © 2001 Autodesk, Inc.

All Rights Reserved

AUTODESK, INC. MAKES NO WARRANTY, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, REGARDING THESE MATERIALS AND MAKES SUCH MATERIALS AVAILABLE SOLELY ON AN "AS-IS" BASIS.

IN NO EVENT SHALL AUTODESK, INC. BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH OR ARISING OUT OF PURCHASE OR USE OF THESE MATERIALS. THE SOLE AND EXCLUSIVE LIABILITY TO AUTODESK, INC., REGARDLESS OF THE FORM OF ACTION, SHALL NOT EXCEED THE PURCHASE PRICE OF THE MATERIALS DESCRIBED HEREIN.

Autodesk, Inc. reserves the right to revise and improve its products as it sees fit. This publication describes the state of this product at the time of its publication, and may not reflect the product at all times in the future.

Autodesk Trademarks

The following are registered trademarks of Autodesk, Inc., in the USA and/or other countries: 3D Plan, 3D Props, 3D Studio, 3D Studio MAX, 3D Studio VIZ, 3DSurfer, ActiveShapes, ActiveShapes (logo), Actrix, ADE, ADI, Advanced Modeling Extension, AEC Authority (logo), AEC-X, AME, Animator Pro, Animator Studio, ATC, AUGI, AutoCAD, AutoCAD Data Extension, AutoCAD Development System, AutoCAD LT, AutoCAD Map, Autodesk, Autodesk Animator, Autodesk (logo), Autodesk MapGuide, Autodesk University, Autodesk View, Autodesk WalkThrough, Autodesk World, AutoLISP, AutoShade, AutoSketch, AutoSurf, AutoVision, Biped, bringing information down to earth, CAD Overlay, Character Studio, Design Companion, Design Your World, Design Your World (logo), Drafrix, Education by Design, Generic, Generic 3D Drafting, Generic CADD, Generic Software, Geodyssey, Heidi, HOOPS, Hyperwire, Inside Track, Kinetix, MaterialSpec, Mechanical Desktop, Multimedia Explorer, NAAUG, ObjectARX, Office Series, Opus, PeopleTracker, Physique, Planix, Powered with Autodesk Technology, Powered with Autodesk Technology (logo), RadioRay, Rastation, Softdesk, Softdesk (logo), Solution 3000, Tech Talk, Texture Universe, The AEC Authority, The Auto Architect, TinkerTech, VISION, *WHIPI*, *WHIPI* (logo), Woodbourne, WorkCenter, and World-Creating Toolkit.

The following are trademarks of Autodesk, Inc., in the USA and/or other countries: 3D on the PC, 3ds max, ACAD, Advanced User Interface, AEC Office, AME Link, Animation Partner, Animation Pro Player, A Studio in Every Computer, ATLAST, Auto-Architect, AutoCAD Architectural Desktop, AutoCAD Architectural Desktop Learning Assistance, AutoCAD Learning Assistance, AutoCAD LT Learning Assistance, AutoCAD Simulator, AutoCAD SQL Extension, AutoCAD SQL Interface, Autodesk Animator Clips, Autodesk Animator Theatre, Autodesk Device Interface, Autodesk Inventor, Autodesk PhotoEDIT, Autodesk Point A (logo), Autodesk Software Developer's Kit, Autodesk Streamline, Autodesk View DwgX, AutoFlix, AutoPAD, AutoSnap, AutoTrack, Built with ObjectARX (logo), ClearScale, Colour Warper, Combustion, Concept Studio, Content Explorer, cornerStone Toolkit, Dancing Baby (image), Design 2000 (logo), DesignCenter, Design Doctor, Designer's Toolkit, DesignProf, DesignServer, DWG Linking, DWG Unplugged, DXF, Extending the Design Team, FLI, FLIC, GDX Driver, Generic 3D, gmax, Heads-up Design, Home Series, i-drop, Kinetix (logo), Lightscape, ObjectDBX, onscreen onair online, Ooga-Chaka, Photo Landscape, Photoscape, Plugs and Sockets, PolarSnap, Pro Landscape, QuickCAD, Real-Time Roto, Render Queue, SchoolBox, Simply Smarter Diagramming, SketchTools, Sparks, Suddenly Everything Clicks, Supportdesk, The Dancing Baby, Transform Ideas Into Reality, Visual LISP, Visual Syllabus, VIZable, Volo, and Where Design Connects.

Third Party Trademarks

All other brand names, product names or trademarks belong to their respective holders.

Third Party Software Program Credits

ACIS Copyright © 1989-2001 Spatial Corp.

Copyright © 1997 Microsoft Corporation. All rights reserved.

International CorrectSpell™ Spelling Correction System © 1995 by Lernout & Hauspie Speech Products, N.V. All rights reserved.

InstallShield™ 3.0. Copyright © 1997 InstallShield Software Corporation. All rights reserved.

Portions Copyright © 1991-1996 Arthur D. Applegate. All rights reserved.

Portions of this software are based on the work of the Independent JPEG Group.

Typefaces from the Bitstream ® typeface library copyright 1992.

Typefaces from Payne Loving Trust © 1996. All rights reserved.

GOVERNMENT USE

Use, duplication, or disclosure by the U. S. Government is subject to restrictions as set forth in FAR 12.212 (Commercial Computer Software-Restricted Rights) and DFAR 227.7202 (Rights in Technical Data and Computer Software), as applicable.

Contents

Introduction	1
Overview of AutoCAD ActiveX Technology	2
Overview of AutoCAD ActiveX Objects	3
Overview of AutoCAD Visual Basic for Applications (VBA) Interface	3
How VBA Is Implemented in AutoCAD	4
Dependencies and Restrictions When Using AutoCAD VBA	5
Examining the Strengths of AutoCAD ActiveX and VBA Together	5
How This Guide Is Organized	6
Conventions Used in This Guide.	6
Typographical Conventions	7
Finding Sample Code	8
Running the Example Code in This Guide	8
Reviewing the Sample Applications	9
 Chapter 1 Getting Started with VBA	 11
Understanding Embedded and Global VBA Projects.	12
Organizing Your Projects with the VBA Manager.	13
Loading an Existing Project	13
Unloading a Project	14
Embedding a Project into a Drawing.	14
Extracting a Project from a Drawing.	15
Creating a New Project	15
Saving Your Project	16

Handling Your Macros	16
Using the Macros Dialog Box	16
Running a Macro	18
Editing a Macro	18
Stepping into a Macro	18
Setting the Project Options.	19
Editing Your Projects with the VBA IDE	20
Viewing Project Information	20
Defining the Components in a Project	21
Importing Existing Components	22
Editing Components.	23
Naming Your Project.	25
Saving Your Project	26
Referencing Other VBA Projects	26
Setting the VBA IDE Options	28
Performing an Introductory Exercise	29
Getting More Information	30
Reviewing AutoCAD VBA Project Terms	30
Reviewing the AutoCAD VBA Commands	31

Chapter 2 Understanding ActiveX Automation Basics 33

Understanding the AutoCAD Object Model	34
A Brief Look at the Application Object	36
A Brief Look at the Document Object	36
A Brief Look at the Collection Objects	38
A Brief Look at the Graphical and Nongraphical Objects	38
A Brief Look at the Preferences, Plot, and Utility Objects	39
Accessing the Object Hierarchy	40
Referencing Objects in the Object Hierarchy	40
Accessing the Application Object	41
Working with the Collection Objects	42
Accessing a Collection	43
Adding a New Member to a Collection Object.	43
Iterating through a Collection Object	43
Deleting a Member of a Collection Object	44
Understanding Properties and Methods	45
Understanding Parent Objects	45
Locating the Type Library	45

Using Variants in Methods and Properties	46
What Is a Variant?	46
Using Variants for Array Data	46
Converting Arrays to Variants	47
Interpreting Variant Arrays	48
Using Other Programming Languages	48
Converting the VBA Code to VB	49
 Chapter 3 Controlling the AutoCAD Environment	53
Opening, Saving, and Closing Drawings	54
Setting AutoCAD Preferences	55
Database Preferences	56
Controlling the Application Window	57
Controlling the Drawing Windows	58
Positioning and Sizing the Document Window	58
Using Zoom	59
Using Named Views	62
Using Tiled Viewports	63
Updating the Geometry in the Document Window	66
Resetting Active Objects	66
Setting and Returning System Variables	67
Drawing with Precision	67
Adjusting Snap and Grid Alignment	68
Using Ortho Mode	69
Drawing Construction Lines	69
Calculating Points and Values	72
Calculating Areas	73
Prompting for User Input	75
GetString Method	75
GetPoint Method	76
GetKeyword Method	76
Controlling User Input	77
Accessing the AutoCAD Command Line	78
Working with No Documents Open	79
Importing Other File Formats	80
Exporting to Other File Formats	80

Chapter 4	Creating and Editing AutoCAD Entities.	83
Creating Objects		84
Determining the Container Object		84
Creating Lines		85
Creating Curved Objects		86
Creating Point Objects		87
Creating Solid-Filled Areas		88
Working with Regions		89
Creating Hatches		92
Working with Selection Sets		95
Creating a Selection Set		96
Adding Objects to a Selection Set		96
Defining Rules for Selection Sets		97
Displaying Information About a Selection Set		105
Removing Objects from a Selection Set		106
Editing Objects		107
Working with Named Objects		107
Copying Objects		108
Offsetting Objects		110
Mirroring Objects		112
Arraying Objects		113
Moving Objects		116
Rotating Objects		117
Deleting Objects		119
Scaling Objects		119
Transforming Objects		121
Extending and Trimming Objects		124
Exploding Objects		124
Editing Polylines		126
Editing Splines		127
Editing Hatches		129
Using Layers, Colors, and Linetypes		133
Working with Layers		133
Working with Colors		139
Working with Linetypes		140
Assigning Layers, Colors, and Linetypes to Objects		143
Saving and Restoring Layer Settings		146
Understanding How AutoCAD Saves Layer Settings		147
Using the LayerStateManager to Manage Layer Settings		148

Adding Text to Drawings	153
Working with Text Styles	153
Using Line Text (Text)	159
Using Multiline Text (Mtext)	163
Using Unicode Characters, Control Codes, and Special Characters	168
Substituting Fonts	169
Checking Spelling	170
 Chapter 5 Dimensioning and Tolerancing	171
Reviewing Dimensioning Concepts	172
Looking at the Parts of a Dimension	173
Defining the Dimension System Variables	173
Setting Dimension Text Styles	174
Understanding Leader Lines	174
Understanding Associative Dimensions	175
Creating Dimensions	175
Creating Linear Dimensions	176
Creating Radial Dimensions	177
Creating Angular Dimensions	178
Creating Ordinate Dimensions	179
Editing Dimensions	181
Overriding Dimension Text	182
Working with Dimension Styles	182
Creating, Modifying, and Copying Dimension Styles	183
Overriding the Dimension Style	184
Dimensioning in Model Space and Paper Space	189
Creating Leaders and Annotation	189
Creating a Leader Line	190
Adding the Annotation to a Leader Line	191
Leader Associativity	191
Editing Leader Associativity	192
Editing Leaders	192
Using Geometric Tolerances	193
Creating Geometric Tolerances	193
Editing Tolerances	194

Chapter 6	Customizing Toolbars and Menus	195
	Understanding the MenuBar and MenuGroups Collections	196
	Loading Menu Groups	198
	Creating New Menu Groups	199
	Changing the Menu Bar	200
	Inserting Menus in the Menu Bar.	200
	Removing Menus from the Menu Bar	201
	Rearranging Menu Items on the Menu Bar	202
	Creating and Editing Pull-Down and Shortcut Menus	202
	Creating New Menus.	203
	Adding New Menu Items to a Menu	204
	Adding Separators to a Menu	206
	Assigning an Accelerator Key to a Menu Item	206
	Creating Cascading Submenus	207
	Deleting Menu Items from a Menu	209
	Exploring the Properties of Menu Items.	209
	Creating and Editing Toolbars	212
	Creating New Toolbars	212
	Adding New Toolbar Buttons to a Toolbar	213
	Adding Separators to a Toolbar	215
	Defining the Toolbar Button Image	215
	Creating Flyout Toolbars	217
	Floating and Docking Toolbars	218
	Deleting Toolbar Buttons from a Toolbar	220
	Exploring the Properties of Toolbar Items	220
	Creating Macros	222
	Macro Characters Mapped to ASCII Equivalents	222
	Macro Termination	224
	Pausing for User Input	225
	Canceling a Command	226
	Macro Repetition	226
	Use of Single Object Selection Mode.	227
	Creating Status-Line Help for Menu Items and Toolbar Items.	227
	Adding Entries to the Right-Click Menu	228
Chapter 7	Using Events	231
	Understanding the Events in AutoCAD.	232
	Guidelines for Writing Event Handlers	232
	Handling Application Level Events	234
	Enabling Application Level Events	235

Handling Document Level Events	237
Enabling Document Level Events in Environments Other Than VBA.	240
Coding Document Level Events in Environments Other Than VBA.	241
Coding Document Level Events in VBA	241
Handling Object Level Events	242
Enabling the Object Level Event.	242
 Chapter 8 Working in Three-Dimensional Space	245
Specifying 3D Coordinates	246
Defining a User Coordinate System	247
Converting Coordinates	249
Creating 3D Objects	251
Creating Wireframes	251
Creating Meshes.	252
Creating a Polyface Mesh	253
Creating Solids	255
Editing in 3D	256
Rotating in 3D	256
Arraying in 3D	258
Mirroring in 3D	258
Editing 3D Solids	260
 Chapter 9 Defining Layouts and Plotting	263
Understanding Model Space and Paper Space	264
Understanding Layouts	264
Understanding the Relationship between Layouts and Blocks	265
Understanding Plot Configurations	265
Determining Layout Settings.	265
Understanding Viewports	267
Working with Floating Viewports	268
Switching to a Paper Space Layout	270
Switching to the Model Space Layout	270
Creating Paper Space Viewports	271
Changing Viewport Views and Content	273
Scaling Views Relative to Paper Space	274
Scaling Pattern Linetypes in Paper Space	275
Hiding Lines in Plotted Viewports	276

Plotting Your Drawing	276
Performing Basic Plotting	277
Plotting from Model Space	278
Plotting from Paper Space	279
Chapter 10 Advanced Drawing and Organizational Techniques.	281
Working with Raster Images	282
Attaching and Scaling a Raster Image	282
Managing Raster Images	283
Modifying Images and Image Boundaries	284
Clipping Images	287
Using Blocks and Attributes	289
Working with Blocks.	289
Working with Attributes	294
Using External References	301
Updating Xrefs	302
Attaching Xrefs	302
Detaching Xrefs	304
Reloading Xrefs	305
Unloading Xrefs	305
Binding Xrefs	306
Clipping Blocks and Xrefs	308
Demand Loading and Maximizing Xref Performance	308
Assigning and Retrieving Extended Data	309
Chapter 11 Developing Applications with VBA	311
More VBA Terminology.	312
Working with Forms in VBA	312
Designing in Design Mode, Running in Run Mode	313
Adding Controls to a Form.	314
Displaying and Hiding Forms	315
Loading and Unloading Forms	316
Designing Your Application for Use with Modal Forms	317
Handling Errors	318
Defining Application Error Types.	318
Trapping Runtime Errors	319
Responding to Trapped Errors	321
Responding to AutoCAD User Input Errors.	321
Encrypting VBA Code Modules	321
Running a VBA Macro from a Toolbar or Menu	322
Automatically Loading a VBA Project	322
Automatically Running a VBA Macro	322
Automatically Opening the VBA IDE Whenever a Project Is Loaded	323

Working in a Zero Document State	323
Distributing Your Application	324
Distributing Visual Basic Applications	324
Chapter 12 Interacting with Other Applications and Windows APIs	325
Interacting with Visual LISP Applications	326
Interacting with Other Windows Applications	326
Referencing the ActiveX Object Library of Other Applications	327
Creating an Instance of the Other Application	328
Programming with Objects from Other Applications	329
Accessing Windows APIs from VBA	332
Appendix A Visual LISP and ActiveX/VBA Comparison	335
AutoLISP and ActiveX/VBA Comparison	336
Appendix B Migrating from AutoCAD Release 14.01	347
New Items	348
Changed Items	360
Changes to the Preferences Object	362
Removed Items	363
Index	365

Introduction

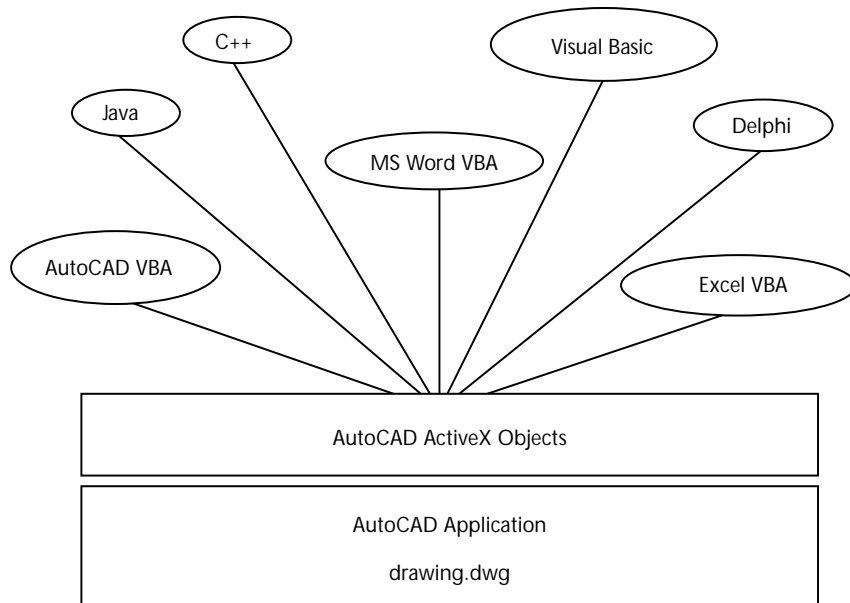
This introduction describes the concept of exposing AutoCAD objects through an ActiveX[®] interface and programming those objects using the Visual Basic for Applications programming environment. Also included is an introduction to all the documentation and sample code provided for AutoCAD ActiveX and VBA.

In this chapter

- Overview of AutoCAD ActiveX Technology
- Overview of AutoCAD Visual Basic for Applications (VBA) Interface
- Examining the Strengths of AutoCAD ActiveX and VBA Together
- How This Guide Is Organized
- Conventions Used in This Guide
- Finding Sample Code

Overview of AutoCAD ActiveX Technology

AutoCAD ActiveX provides a mechanism to manipulate AutoCAD programmatically from within or outside AutoCAD. It does this by exposing AutoCAD objects to the “outside world.” Once these objects are exposed, they can be accessed by many different programming languages and environments and by other applications such as Microsoft® Word VBA or Excel VBA.



There are two advantages to implementing an ActiveX interface for AutoCAD:

- Programmatic access to AutoCAD drawings is opened up to many more programming environments. Before ActiveX Automation, developers were limited to an AutoLISP or C++ interface.
- Sharing data with other Windows® applications, such as Microsoft Excel® and Word®, is made dramatically easier.

Overview of AutoCAD ActiveX Objects

An object is the main building block of any ActiveX application. Each exposed object represents a precise part of AutoCAD. There are many different types of objects in the AutoCAD ActiveX interface. For example

- Graphical objects such as lines, arcs, text, and dimensions are objects.
- Style settings such as linetypes and dimension styles are objects.
- Organizational structures such as layers, groups, and blocks are objects.
- The drawing displays such as view and viewport are objects.
- Even the drawing and the AutoCAD application are considered objects.

Overview of AutoCAD Visual Basic for Applications (VBA) Interface

Microsoft VBA is an object-oriented programming environment designed to provide rich development capabilities similar to those of Visual Basic (VB). The main difference between VBA and VB is that VBA runs in the same process space as AutoCAD, providing an AutoCAD-intelligent and very fast programming environment.

VBA also provides application integration with other VBA-enabled applications. This means that AutoCAD, using other application object libraries, can be an Automation controller for other applications such as Microsoft Word or Excel.

The standalone development editions of Visual Basic, which must be purchased separately, complement AutoCAD VBA with additional components, such as an external database engine and report-writing capabilities.

There are four advantages to implementing VBA for AutoCAD:

- The Visual Basic programming environment is easy to learn and use.
- VBA runs in-process with AutoCAD. This translates to very fast program execution.
- Dialog construction is quick and effective. This allows developers to prototype applications and quickly receive feedback on designs.
- Projects can be standalone or imbedded in drawings. This choice allows developers great flexibility in the distribution of their applications.

How VBA Is Implemented in AutoCAD

VBA sends messages to AutoCAD by the AutoCAD ActiveX Automation interface. AutoCAD VBA permits the VBA environment to run simultaneously with AutoCAD and provides programmatic control of AutoCAD through the ActiveX Automation interface. This coupling of AutoCAD, ActiveX Automation, and VBA provides an extremely powerful interface not only for manipulating AutoCAD objects, but for sending data to or retrieving data from other applications.

There are three fundamental elements that define ActiveX and VBA programming in AutoCAD. The first is AutoCAD itself, which has a rich set of objects that encapsulates AutoCAD entities, data, and commands. Because AutoCAD was designed as an open-architecture application with multiple levels of interface, familiarity with AutoCAD programmability is highly desirable in order to use VBA effectively. If you've used AutoLISP® to control AutoCAD programmatically, you already have a good understanding of the AutoCAD facilities. However, you will find the VBA object-based approach to be quite different from that of AutoLISP.

The second element is the AutoCAD ActiveX Automation interface, which establishes messages (communication) with AutoCAD objects. Programming in VBA requires a fundamental understanding of ActiveX Automation. A description of the AutoCAD ActiveX Automation interface can be found in the *ActiveX and VBA Reference*. Even the experienced VB programmer will find the AutoCAD ActiveX Automation interface invaluable for understanding and developing AutoCAD VBA applications.

The third element is the VBA programming environment which has its own set of objects, keywords, constants, and so forth that provides program flow, control, debugging, and execution. Microsoft's own extensive online help for VBA is included with the AutoCAD VBA and is accessible from the VBA IDE by any of the following methods:

- Pressing F1 on the keyboard
- Choosing Help from the VBA IDE menus
- Clicking the Question Mark icon on the VBA IDE toolbar

Dependencies and Restrictions When Using AutoCAD VBA

To ensure the proper functionality of AutoCAD ActiveX and VBA you must comply with a few system dependencies.

Operating System Dependencies

Windows NT[®] 4.0 It is highly recommended that Service Pack 3 for Windows NT 4.0 be installed to run AutoCAD ActiveX and VBA.

Windows[®] 95, or Windows 98 No special requirements from Microsoft.

Installing, Reinstalling, or Uninstalling Microsoft Office or Other VBA Applications

If you install, reinstall, or uninstall Microsoft Office or other VBA applications after installing AutoCAD, you will need to reinstall AutoCAD. It is highly recommended that after any installation of AutoCAD you reboot your system.

Examining the Strengths of AutoCAD ActiveX and VBA Together

The AutoCAD ActiveX/VBA interface represents several advantages over other AutoCAD API environments:

- *Speed*

Running in-process with VBA, ActiveX applications are faster than either AutoLISP or ADS applications.

- *Ease of Use*

The programming language and development environment are easy to use and come installed with AutoCAD.

- *Windows Interoperability*

ActiveX and VBA are designed to be used with other Windows applications and provide an excellent path for communication of information across applications.

- *Rapid Prototyping*

The rapid interface development of VBA provides the perfect environment for prototyping applications, even if those applications will eventually be developed in another language.

- *Programmer Base*

There are millions of Visual Basic programmers around the world. AutoCAD ActiveX and VBA technology open up AutoCAD customization and application development to these programmers and the many more who will learn Visual Basic in the future.

How This Guide Is Organized

This guide provides information regarding the development of ActiveX and VBA applications for use with AutoCAD. Information specific to developing applications using VBA can be found in chapter 1, “Getting Started with VBA,” and chapter 11, “Developing Applications with VBA.” Programmers using ActiveX from a development environment other than VBA can skip these two chapters. However, be aware that all of the example code in this guide is presented in VBA.

Information on migrating from AutoCAD Release 14.01 to AutoCAD 2002 can be found in appendix B, “Migrating from AutoCAD Release 14.01.” This information summarizes the changes to AutoCAD ActiveX and VBA since these features first appeared in AutoCAD.

Conventions Used in This Guide

This guide assumes you have a working knowledge of the Visual Basic programming language, and does not attempt to duplicate or replace the abundance of documentation available on Visual Basic. If you need more information on the Visual Basic language or development environment usage, see the Visual Basic for Applications Help file developed by Microsoft, available from the Help menu in the interactive development environment (IDE).

Typographical Conventions

To orient you to AutoCAD features, specific terms are set in typefaces that distinguish them from the body text. Throughout AutoCAD documentation, the following conventions are used.

Typographical conventions	
Text element	Example
AutoCAD commands	ADDCENTER, DBCONNECT, SAVE
AutoCAD system variables	DIMBLK, DWGNAME, LWSCALE
AutoCAD named objects, such as linetypes and styles	DASHDOT, STANDARD
Prompts	Select object to trim or [Project/Edge/Undo]:
Instructions after prompt sequences	Select objects: <i>Use an object selection method</i>
File names	<i>acad2000.cfg, Readme file</i>
File name extensions	<i>.dwg file name extension</i>
Folder or directory names	<i>Sample folder, c:\ACAD2000\support</i>
Text you enter	At the Command prompt, enter shape
Keys you press on the keyboard	CTRL, F10, ESC, ENTER
Keys you press simultaneously on the keyboard	CTRL + C
ActiveX argument names, constants, sample code, and VBA keywords	To create a new button as a flyout use the AddToolBarButton method and set the Fl youtButton argument to TRUE. ci rcl e. Col or = acRed
AutoLISP variable names, sample code, and text in ASCII files	The variable pi is preset to a value of pi ***P0P1
AutoLISP and DIESEL function names	command ads_command()
AutoLISP Formal arguments specified in function definitions	The <i>string</i> and <i>mode</i> arguments

Finding Sample Code

This manual and the *ActiveX and VBA Reference* together contain over 800 example VBA subroutines that demonstrate the usage of ActiveX methods, properties, and events.

There are also many sample applications provided in the AutoCAD *Sample* directory. These sample applications demonstrate a wide range of functionality, from extracting AutoCAD drawing data into Microsoft Excel spreadsheets to drawing and performing stress analysis on an electrical transmission tower. These samples will show you how to combine the versatility of the Visual Basic for Applications programming environment together with the power of AutoCAD ActiveX interface to create customized applications.

Running the Example Code in This Guide

All of the example code in the *ActiveX and VBA Developer's Guide* and *ActiveX and VBA Reference* can be copied from the help files, pasted directly into the AutoCAD VBA environment, and then executed with one requirement: the current active drawing in AutoCAD must be a blank drawing open to model space. Additionally, the code in these manuals can be found in the *SampleCode.dvb* and *Events.dvb* files in the *Sample* directory, which is in your AutoCAD installation directory.

To run the examples

- 1 Copy the example from the help file into an empty VBA code module.
- 2 Verify that AutoCAD has a blank drawing open to model space.
- 3 Open the Macros dialog box by entering the command VBARUN.
- 4 Choose the macro and press Run.

More information on running macros and the Macros dialog box is available in "Running a Macro" on page 18.

Reviewing the Sample Applications

The following table provides the name, description, and location of the main source code file for each sample application in the *Sample* directory. Many sample applications have support files that can be found in the same directory as the main source code file. There is also a *readme.txt* file in the same directory as the main source code file that describes the application and how to run it.

ActiveX and VBA sample applications

Name	Description	Location
IBeam 3D	Creates a 3D solid IBeam, and dynamically resizes it.	<i>/Sample/VBA/ibeam3d.dvb</i>
Map to Globe	Generates 3D polylines on a sphere from the original 2D polylines.	<i>/Sample/VBA/Map2Globe.dwg</i>
Menu Customization	Demonstrates using the MenuGroup and MenuBar objects.	<i>/Sample/VBA/Menu.dvb</i>
Object Tracker	Uses Xrecord data to track object modifications.	<i>/Sample/VBA/ObjectTracker.dvb</i>
Save As R12	Saves an AutoCAD 2002 drawing file into the AutoCAD Release 12 drawing format.	<i>/Sample/VBA/SaveAsR12.dvb</i>
Tower	Draws an electrical transmission tower and performs a stress analysis on it.	<i>/Sample/VBA/Tower.dwg</i>
Text Height	Globally changes text height for all text in a drawing.	<i>/Sample/VBA/txtht.dvb</i>
External Call	A VBA macro that calls an external Visual Basic 6 function in a registered DLL.	<i>/Sample/ActiveX/ExtrnCall/ExternalCall.dvb</i>
Facility	Demonstrates connecting AutoCAD to a database.	<i>/Sample/ActiveX/Facility/Setup/Setup.exe</i>
VBA IDE Customization	Creates a new toolbar in the VBA IDE allowing you to load a project, and bring up the VBA Manager, VBA Macros, and VBA Options dialog from the IDE.	<i>/Sample/VBA/VBAIDEMenu/acad.dvb</i>

ActiveX and VBA sample applications (continued)

Name	Description	Location
Extract Attributes	This Excel macro extracts AutoCAD block data and deposits the data in a spreadsheet.	<i>/Sample/ActiveX/ExtAttr/ExtAttr.xls</i>
Attribute Text	This AutoCAD macro extracts attribute data to Microsoft Word document, Excel spreadsheet, and Chart.	<i>/Sample/VBA/atttext.dvb</i>
Excel Link	This AutoCAD macro demonstrates how to pass data from AutoCAD to Excel and then back to AutoCAD again.	<i>/Sample/VBA/ExcelLink.dvb</i>
Block Replace	Replaces inserted blocks in drawing with a different block definition.	<i>/Sample/VBA/BlockReplace.dvb</i>
Change Poly Width	Globally changes the widths of all polylines in the drawing.	<i>/Sample/VBA/chplywid.dvb</i>
Draw Centerline	Draws centerlines for arcs, ellipses, and circles.	<i>/Sample/VBA/cntrline.dvb</i>
Draw Line	Demonstrates how to draw a line from a VBA form.	<i>/Sample/VBA/drawline.dvb</i>
Example Code	All example code, except for the events examples, from the <i>ActiveX and VBA Developer's Guide</i> and the <i>ActiveX and VBA Reference</i> .	<i>/Sample/VBA/Example_Code.dvb</i>
Example Events	The events examples from the <i>ActiveX and VBA Reference</i> .	<i>/Sample/VBA/Example_Events.dvb</i>
Miscellaneous	Demonstrates various Automation APIs using a dialog interface.	<i>/Sample/VBA/acad_cg.dvb</i>

Getting Started with VBA

1

This chapter introduces you to AutoCAD VBA projects and the VBA IDE. Although most VBA environments are similar in behavior, the AutoCAD VBA IDE has some unique features. There are also several AutoCAD commands that can be used to load projects, run projects, or open the VBA IDE. This chapter defines the use of VBA projects, VBA commands, and the VBA IDE in general.

In this chapter

- Understanding Embedded and Global VBA Projects
- Organizing Your Projects with the VBA Manager
- Handling Your Macros
- Editing Your Projects with the VBA IDE
- Performing an Introductory Exercise
- Getting More Information
- Reviewing AutoCAD VBA Project Terms
- Reviewing the AutoCAD VBA Commands

Understanding Embedded and Global VBA Projects

An AutoCAD VBA project is a collection of code modules, class modules, and forms that work together to perform a given function. Projects can be stored within an AutoCAD drawing, or as a separate file.

Embedded projects are stored within an AutoCAD drawing. These projects are automatically loaded whenever the drawing in which they are contained is opened in AutoCAD, making the distribution of projects very convenient. Embedded projects are limited and not able to open or close AutoCAD drawings because they function only within the document where they reside. Users of embedded projects are no longer required to find and load project files before they run a program. A time log that is triggered when the drawing is opened is an example of a project embedded in a drawing. With this macro users can log in and record the length of time they worked on the drawing. The user does not have to remember to load the project before opening the drawing; it simply is done automatically.

Global projects are stored in separate files and are more versatile because they can work in, open, and close any AutoCAD drawing, but are not automatically loaded when a drawing is opened. Users must know which project file contains the macro they need and then load that project file before they can run the macro. However, global projects are easier to share with other users, and they make excellent libraries for common macros. An example of a project you may store in a project file is a macro that collects a bill of materials from many drawings. This macro can be run by an administrator at the end of a work cycle and can collect information from many drawings.

At any given time, users can have both embedded and global projects loaded into their AutoCAD session.

AutoCAD VBA projects are not binary compatible with standalone Visual Basic projects. However, the forms, modules, and classes can be exchanged between projects using the `IMPORT` and `EXPORT` VBA commands in the VBA IDE. For more information on the VBA IDE, see “Editing Your Projects with the VBA IDE” on page 20.

Organizing Your Projects with the VBA Manager

You can view all the VBA projects loaded in the current AutoCAD session by using the VBA Manager. It is an AutoCAD tool that allows you to load, unload, save, create, embed, and extract VBA projects.

To open the VBA Manager

- 1 From the Tools menu choose Macro ► VBA Manager.
- 2 Or, in AutoCAD invoke the **VBAMAN** command.

Loading an Existing Project

When you load a project into AutoCAD, all the public subroutines, also called macros, become available for use. Projects embedded in a drawing are loaded whenever the drawing is opened. Projects stored in DVB files must be loaded explicitly.

Anytime a project is loaded, any other projects that are referenced by the first project will be loaded automatically. Additionally, AutoCAD will automatically load at startup any project file with the name *acad.dvb*.

To load an existing VBA project file

- 1 In the VBA Manager, use the Load option to bring up the Open VBA Project dialog box.
- 2 In the Open VBA Project dialog box, select the project file to open. The VBA Project dialog box will allow you to open only valid DVB files. If you attempt to open a different type of file, you will receive an error message.
- 3 Select Open.

You can also load a project file using one of the following methods:

- Enter the **VBALOAD** command, which opens the Open VBA Project dialog box.
- Drag a DVB file from Windows Explorer and drop it into an open drawing in the AutoCAD window.

Virus Alert

Each time you load a project you are given the option of enabling or disabling the code within that project as a protection against viruses. If you enable the code, viruses in the code can begin executing. If you disable the code, the project will still be loaded, but all code within that project is prevented from running. The virus protection alert is not displayed when you load a project by dragging a DVB file from Windows Explorer and dropping it into an open drawing in the AutoCAD window.

More information on the virus protection is available in “Setting the Project Options” on page 19.

Unloading a Project

Unloading a project frees up memory and keeps the list of loaded projects at a length that is easy to manage.

You cannot unload embedded projects or projects that are referenced by other loaded projects.

To unload a VBA project

- 1 In the VBA Manager, select the project you want to unload.
- 2 Choose Unload.
- 3 Or, use the **VBAUNLOAD** command, which prompts you for the project to be unloaded.

Embedding a Project into a Drawing

When you embed a project you place a copy of the project in the drawing database. The project is then loaded or unloaded whenever the drawing containing it is opened or closed.

A drawing can contain only one embedded project at a time. If a drawing already contains an embedded project you must extract it before a different project can be embedded into the drawing.

To embed a project in an AutoCAD drawing

- 1 Open the VBA Manager and select the project you want to embed.
- 2 Choose Embed.

Extracting a Project from a Drawing

When you extract a project you remove the project from the drawing database and are given the opportunity to save the project in an external project file. If you do not save the file in an external project file, the project data will be deleted.

To extract a project from an AutoCAD drawing

- 1 Open the VBA Manager and select the drawing from which the project is to be extracted.
- 2 Choose Extract.
- 3 If you want to save the project information in an external project file, choose Yes to the prompt “Do you want to export the VBA project before removing it?” The Save As dialog box will be displayed, allowing you to save the file.

If you do not want to save the project information in an external file, choose No to the prompt “Do you want to export the VBA project before removing it?” The project information will be removed from the drawing and will not be saved.

Creating a New Project

New projects are created as unsaved global projects. Once a project has been created, you can then embed the project in a drawing, or save the project out to a project file.

To create a new VBA project

- 1 Open the VBA Manager.
- 2 Choose New.

A new project will be created with the default name of *ACADProject*. To change the project name you must go into the VBA IDE. For more information on naming your project in the VBA IDE, see “Naming Your Project” on page 25.

Saving Your Project

Embedded projects are saved whenever the drawing is saved. Global projects must be saved using the VBA Manager or the VBA IDE.

To save your project using the VBA Manager

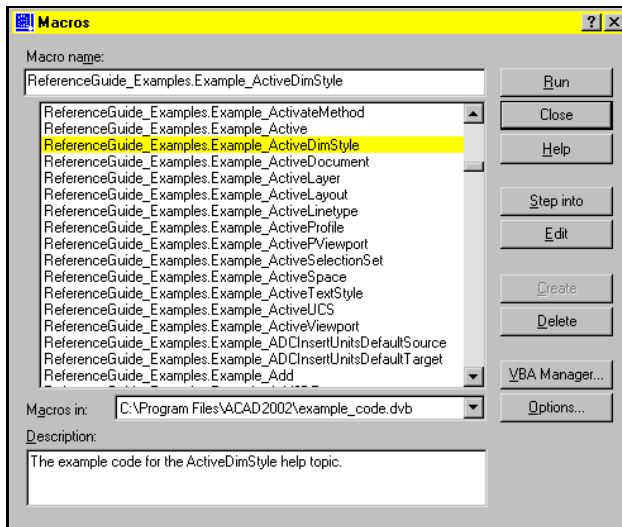
- 1 Open the VBA Manager and select the project to be saved.
- 2 Choose Save As. The Save As dialog box will open.
- 3 Select the file name for the project to be saved in.
- 4 Choose Save.

Handling Your Macros

A macro is a public (executable) subroutine. Each project usually contains at least one macro.

Using the Macros Dialog Box

The Macros dialog box allows you to run, edit, delete, and create macros as well as set the VBA project options. Open the Macros dialog box from the AutoCad Tools menu by choosing Macro ► Macros, or issue **VBARUN** at the AutoCAD Command prompt.



The names of all macros in the valid range are displayed in this dialog box. You can change the valid range by using the Macros In drop-down list. This list specifies the projects or drawings whose macros are displayed. You can choose to display the macros in

- All drawings and projects
- All drawings
- All projects
- Any individual drawing currently open in AutoCAD
- Any individual project currently loaded in AutoCAD

By limiting the valid range you can control how many macro names appear in the list. This will help you in the cases when many macros are available in the loaded drawings and projects.

To create a new macro

- 1 Open the Macros dialog box and enter the name for the new macro.
- 2 In the Macros In drop-down list, select a project to create the new macro in.
- 3 Choose Create.

If a macro with the specified name already exists, you will be asked if you want to replace the existing macro.

If you select Yes at the prompt, the code in the existing macro will be deleted and a new, empty macro will be created with the specified name.

If you select No at the prompt, you will be returned to the Macros dialog box to enter a new name for the macro.

If you select Cancel at the prompt, the Macros dialog box will be dismissed and no new macro will be created.

To delete a macro

- 1 Open the Macros dialog box and select the macro to delete.
- 2 Choose Delete. You will be prompted to confirm the delete.
- 3 At the prompt, choose Yes to delete the macro, or No to cancel the delete.

Running a Macro

Running a macro executes the macro code within the context of the current AutoCAD session. The current active drawing is considered to be the open drawing that has the focus when macro execution begins. All VBA references to the `ThisDrawing` object will refer to the current active drawing for macros in global projects. For macros in embedded projects, the `ThisDrawing` object always refers to the drawing in which the macro is embedded.

To run a macro from the Macros dialog box

- 1 Open the Macros dialog box and select the macro to run.
- 2 Choose Run.

To run a macro from the VBA IDE

- From the Run menu, use the Run Macro menu option.
 - If no macro or form is current, a dialog box will display allowing you to choose the macro to run.
 - If a given macro is current (the cursor is in a procedure), that macro will be executed.

Editing a Macro

Editing a macro will open the VBA IDE with the chosen macro open in the Code window. For more information on editing macros in the VBA IDE see “Editing Your Projects with the VBA IDE” on page 20.

To edit a macro

- 1 Open the Macros dialog box and select the macro to edit.
- 2 Choose Edit.

Stepping into a Macro

Stepping into a macro begins execution of the macro and then halts the execution on the first line of code. The VBA IDE is opened with the chosen macro open in the Code window at the line of execution.

To step into a macro

- 1 In the Macros dialog box, select the macro to step into.
- 2 Choose Step.

Setting the Project Options

There are three options that can be set for AutoCAD VBA projects:

- Enabling Auto Embedding
- Allowing Break on Errors
- Enabling Macro Virus Protection

To set the AutoCAD VBA project options

- 1 From the Tools menu choose Macro ► Macros to open the VBA Macros dialog box.
- 2 From the VBA Macros dialog box, choose Options to open the Options dialog box.
- 3 From the Options dialog box, select the options you want to enable.
- 4 Choose OK.

Enabling Auto Embedding

The auto embed feature automatically creates an embedded VBA project for all drawings when the drawing is opened.

Allowing Break on Errors

This option allows VBA to enter Break mode when an error is encountered. Break mode is a temporary suspension of program execution in the interactive development environment. In Break mode, you can examine, debug, reset, step through, or continue program execution.

When this option is enabled, unhandled errors found during the execution of a VBA macro will suspend the execution of the macro and display the VBA IDE at the point of the error in the macro.

When this option is disabled, untrapped errors found during the execution of a VBA macro will display a message box alerting you to the error, and then end execution of the macro.

Enabling Macro Virus Protection

The virus protection mechanism displays a built-in warning message whenever you open a drawing that may contain macro viruses.

Editing Your Projects with the VBA IDE

Once a project has been loaded into AutoCAD, you can edit the code, forms, and references for that project using the VBA interactive development environment. You can also debug and run projects from the VBA IDE. Once open, the VBA IDE provides access to all loaded projects.

To open the VBA IDE on demand

You can open the VBA IDE from the command line or from the menu bar.

- From the command line, enter **VBAIDE**, or from the Tools menu, choose Macro ► Visual Basic Editor.

To open the VBA IDE automatically on AutoCAD startup

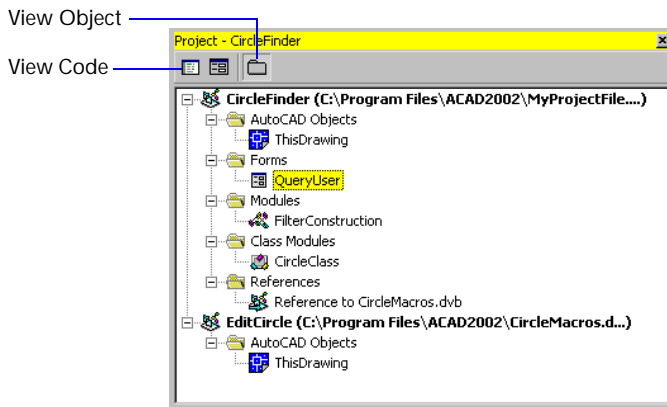
If you want to open the VBA IDE automatically every time you start AutoCAD you will need to include the following line in the *acad.rx* file:

```
acadvba. arx
```

Viewing Project Information

The VBA IDE contains a window called the Project window, which displays a list of all loaded VBA projects. It also displays the code, class, and form modules included in the project, the document associated with the project, all other VBA projects referenced from the project, and the physical location (path) of the project.

The Project window has its own toolbar, which can be used to open various project components for editing. Use the View Code button to open the code for a selected module. Use the View Object button to display selected objects such as forms.



The Project window is visible by default. If it is not visible, select Project window from the View menu, or press CTRL+R.

Defining the Components in a Project

Each project can contain many different components. The different components a project can contain are objects, forms, standard modules, class modules, and references.

Objects

The object component represents the type of object, or document, that the VBA code will access. For AutoCAD VBA projects, this object represents the current AutoCAD drawing.

Forms

The form component contains the custom dialog boxes you constructed for use with your project.

Standard Modules

The code module component contains your generic procedures and functions. A standard module is also referred to as a code module, or as simply a module.

Class Modules

The class module component contains all your own objects, which are defined as classes.

References

The reference component contains all your references to other projects or libraries.

Adding New Components

Adding new components creates a blank component in your project. You can add new modules, forms, and class modules to your project. You are responsible for updating all the properties of the component (such as the name of the component) and for filling in the appropriate code. When naming new components, remember that other developers may want to use your components in future applications. Follow the appropriate naming conventions for your development team.

To add a new component to your project

- 1 In the Project window of the VBA IDE, select the project to which you will be adding the component.
- 2 From the Insert menu, select UserForm, Module, or Class Module to add the new component to your project.

The new component will be added to your project and will appear in the Project window.

Importing Existing Components

Importing allows you to add an existing component to your project. You can import forms, modules, or class modules. Forms are imported as FRM files, modules are imported as BAS files, and class modules are imported as CLS files.

When you import a component file, a copy of the file you are importing is added to the project. The original file is left intact. Changes you make to the imported component do not alter the original component file.

If you import a component with the same name as an existing one, the file is added to your project with a number appended to it.

The imported component will be added to your project and will appear in the Project window. To edit the properties of the component, select that component in the Project window. The properties for the selected component will be listed and can be edited in the Properties window.

To import an existing component to your project

- 1 In the Project window of the VBA IDE, select the project to which you will be adding the component.
- 2 From the File menu, select Import File to open the Import File dialog box.
- 3 From the Import File dialog box, select the file to import and press Open.

Editing Components

You can edit standard modules, class modules, and forms in the VBA IDE. Standard and class modules are edited in a Code window. Forms are edited in the UserForm window using a special toolbox.

You can open as many Code windows as you have modules, so you can easily view the code in different forms or modules, and copy and paste between them.

To edit a component in your project

- 1 In the Project window of the VBA IDE, select the component you want to edit.
- 2 Select the View Code button in the Project window to open a Code window.
- 3 Select the View Object button in the Project window to open a UserForm window and associated toolbox.

To access the code associated with a form

- To access the code associated with a control, double-click on any control in the Form window. The code associated with that control will open in a Code window.

Using the Code Window

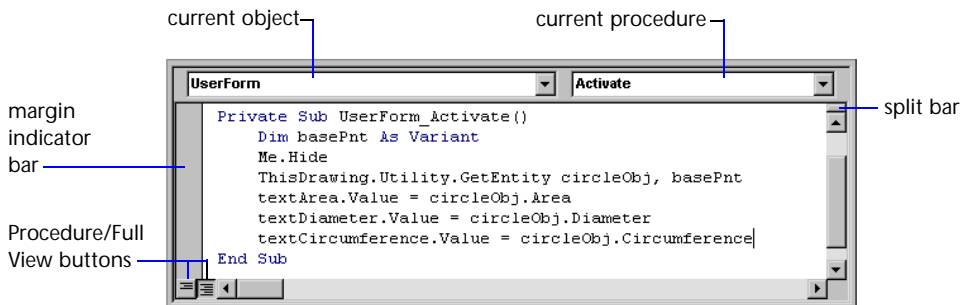
The Code window contains two drop-down lists, a split bar, a margin indicator bar, and the full view and procedure view icons.

The two drop-down lists at the top of the Code window display the current object and procedure. You can move about your project by changing the object or procedure in these drop-down lists.

The split bar on the right side of the Code window allows you to split the window horizontally. Simply drag this bar down to create another window pane. This feature allows you to view two parts of code simultaneously in the same module. To close the pane, drag the split bar back to its original location.

The margin indicator bar is located down the left side of the Code window. It is used to display margin indicators that are used during code editing and debugging.

The full view and procedure view icons are located at the bottom-left corner of the Code window and toggle the display from only one procedure at a time to viewing the entire module at one time.



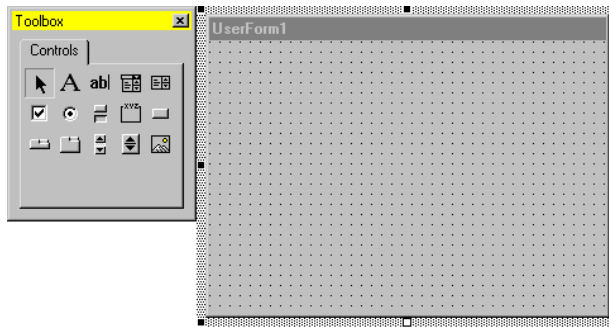
Using the UserForm Window

The UserForm window allows you to create custom dialog boxes in your project.

To add a control simply drag the desired control from the toolbox and place it on the form. You can set your controls to align with the grid of your form from the General tab of the Options dialog box. You can view the form grid and determine the size of the gridlines from the General tab of the Options dialog box. (See "Setting the VBA IDE Options" on page 28 for more information on the Options dialog box.)

Each form you design will automatically have a Maximize, Minimize, and Close button. These buttons have already been implemented for you.

To add code to the control, simply double-click on the control once it has been placed on the form. This will open a Code window for the control.



Naming Your Project

The project name and the name of the *.dvb* file where the project is stored are two different values. You establish the name of the *.dvb* file the project is stored in when you save the project. The project name is set in the Properties window of the VBA IDE.

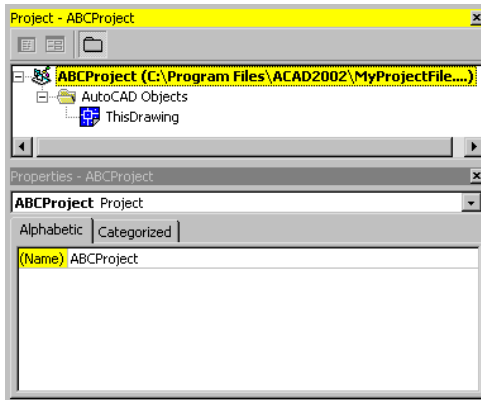
If you do not set the project name and file name, AutoCAD automatically assigns the following default names:

Project name: *ACADProject*

File name: *Project.dvb*

To change the name of a project

- 1 In the Project window of the VBA IDE, select the project to change.
- 2 In the Properties window, edit the Name property for the project.



To change the file name for a project

- 1 In the VBA IDE, select the Save option from the File menu.
- 2 In the Save As dialog box, enter the new name and location for the project file.

Saving Your Project

There is no explicit SAVE command in AutoCAD for VBA projects. Instead, the SAVE command resides in the File menu of the VBA IDE and in the VBA Manager. Any changes to a VBA project will access a standard Save VBA Project dialog box when one of these events occurs:

- You pick the SAVE command from the VBA IDE.
- You choose the Save As option in the VBA Manager.
- Your AutoCAD session is about to end or quit and the VBA project is not saved.

NOTE Before you save a project, it is assigned the default file name *project.dvb*. It is important that you assign a new name to your project file when you save the project. If you save a project with the default file name *project.dvb*, you will no longer be able to create new empty projects. Each time you create a new project, you will actually be loading the saved project called *project.dvb*.

Referencing Other VBA Projects

Referencing one VBA project from another allows developers to share code more easily. Developers can create libraries of commonly used macros and then reference the library when needed. This keeps the shared code centrally

located and supported, while allowing a large number of developers to utilize the code.

Once another project has been successfully referenced, you will notice a new folder in the Projects window of the VBA IDE. This new folder is titled *References* and contains the name of the project referenced.

Once you have referenced a project, you can use any public code or form component in that project.

When a project that references another project is loaded into AutoCAD, the referenced project is automatically loaded into AutoCAD as well. The referenced project cannot be closed until all projects that reference it are closed first.

You cannot make circular references. That is, you cannot reference a project that contains a reference back to the first project. If you accidentally create a circular reference, you will be notified by VBA.

Project referencing is a standard feature of Microsoft VBA. There is no additional work in AutoCAD to extend this functionality. You can find more information on referencing projects in the Microsoft Visual Basic help file. You can open the Microsoft Visual Basic help file from the Help menu in the VBA IDE.

NOTE You cannot reference embedded projects or VBA projects from other applications.

To reference another VBA project

- 1 In the Project window of the VBA IDE, select the project to which you will be adding the reference.
- 2 From the Tools menu, select the References option to open the References dialog box.
- 3 From the References dialog box, press the Browse button to open the Add Reference dialog box.
- 4 From the Add Reference dialog box, select the project file you want to reference and then press the Open button.
- 5 From the Add Reference dialog box, select the OK button to complete the reference addition.

Setting the VBA IDE Options

You can change the characteristics of the VBA IDE using the Options dialog box. To open the Options dialog box, use the Tools menu and select Options.

The Options dialog box contains four tabs: Editor, Editor Format, General, and Docking.

Editor

The Editor tab specifies the Code window and Project window settings.

Code settings include

- Auto Syntax Check
- Require Variable Declaration
- Auto List Member
- Auto Quick Info
- Auto Data Tips
- Auto Indent
- Tab Width

Window settings include

- Drag and Drop Text Editing
- Default to Full Module View
- Procedure Separator Display

Editor Format

The Editor Format tab specifies the appearance of your Visual Basic code.

You can

- Change color of the code
- Change text list items
- Change foreground
- Change background
- Change margin indicators
- Change text font and size
- Display or hide the margin indicator
- Display or hide sample text for your settings

General

The General tab specifies the settings, error handling, and compile settings for your current Visual Basic project.

You can

- Change the grid settings for the form grid
- Display or hide tooltips
- Set the automatic collapse of windows
- Choose to receive state loss notifications
- Determine how errors are handled
- Set the project to compile on demand or perform background compilations

Docking

The Docking tab allows you to choose which windows you want to be dockable.

Performing an Introductory Exercise

Now that you have learned the basics of programming in AutoCAD VBA, let's try creating a simple "Hello World" exercise. In this exercise you will create a new AutoCAD drawing, add a line of text to that drawing, then save the drawing, all from VBA.

Creating the "Hello World" text object

- 1 Open the VBA IDE by entering the following command from the AutoCAD command line:
Command: **VBAIDE**
- 2 Open the Code window by selecting the Code option from the View menu in the VBA IDE.
- 3 Create a new procedure in the project by selecting the Procedure option from the Insert menu in the VBA IDE.
- 4 When prompted for the procedure information, enter a name such as **HelloWorld**. Make sure the Type selected is Sub, and the Scope selected is Public.
- 5 Choose OK.
- 6 Enter the following code (that opens a new drawing) between the lines `Public Sub HelloWorld()` and `End Sub`.
`ThisDrawing.Application.Documents.Add`
- 7 Enter the following code (that creates the text string and defines its insertion location) immediately following the code entered in step 6.

```

Dim insPoint(0 To 2) As Double 'Declare insertion point
Dim textHeight As Double      'Declare text height
Dim textStr As String         'Declare text string
Dim textObj As AcadText       'Declare text object

insPoint(0) = 2               'Set insertion point x coordinate
insPoint(1) = 4               'Set insertion point y coordinate
insPoint(2) = 0               'Set insertion point z coordinate

textHeight = 1                'Set text height to 1.0
textStr = "Hello World!"      'Set the text string

'Create the Text object
Set textObj = ThisDrawing.ModelSpace.AddText _
    (textStr, insPoint, textHeight)

```

8 Enter the code (that saves the drawing) immediately following the code entered in step 7.

```
ThisDrawing.SaveAs("Hello.dwg")
```

9 Run your program by selecting the Run Sub/UserForm option from the Run menu in the VBA IDE.

When the program finishes running, bring the AutoCAD application to the front. You should see your text “Hello World!” visible in your drawing. The drawing name should be *Hello.dwg*.

Getting More Information

More information on the VBA IDE and the Visual Basic programming language is available in the help files provided by Microsoft. To access the Microsoft help files, choose Microsoft Visual Basic Help from the Help menu in the VBA IDE.

Reviewing AutoCAD VBA Project Terms

Global Project

A VBA project stored in a *.dwb* file.

Embedded Project

A VBA project stored in an AutoCAD drawing.

Regular Document	An AutoCAD drawing that does not contain VBA embedded projects.
Smart Document	An AutoCAD drawing that contains one or more VBA embedded projects.
Current Project	The project currently selected in the VBA IDE.
ThisDrawing	ThisDrawing is a VBA programming term used to represent the current drawing. For global projects, ThisDrawing always refers to the active document in AutoCAD. For embedded projects, ThisDrawing always refers to the document containing the project.
VBA IDE	The VBA interactive development environment. This application allows you to edit the code and forms in your project, or copy code and forms from other projects. It also allows you to set references to other application Object Models.
VBA Manager	The VBA Manager allows you to manage your projects. You can create, delete, embed, or extract projects. You can also view which projects, if any, are embedded in an open drawing.
Macros Dialog Box	The Macros dialog box allows you to run, delete, and create new macros, and provides access to the VBA project options.

Reviewing the AutoCAD VBA Commands

VBAIDE	Brings up the VBA IDE. The VBA IDE allows you to edit, run, and debug programs interactively. Although the VBA IDE is invoked only when AutoCAD is running, it can be minimized, opened, and closed independent of the AutoCAD Application window.
VBALOAD	Loads a VBA project into the current AutoCAD session.
VBARUN	Runs a VBA macro from the Macros dialog box or from the AutoCAD command line.

VBAUNLOAD	Unloads a VBA project from the current AutoCAD session. If the VBA project is modified but not saved, the user is asked to save it with the Save Project dialog box (or command line equivalent).
VBAMAN	Displays the VBA Manager allowing you to view, create, load, close, embed, and extract projects.
VBASTMT	Executes a VBA statement from the AutoCAD command line.

Understanding ActiveX Automation Basics

2

To use AutoCAD ActiveX Automation effectively you should be familiar with the AutoCAD entities, objects, and features relating to the type of application you are developing. The greater your knowledge of an object's graphical and nongraphical properties, the easier it is for you to manipulate them through AutoCAD ActiveX Automation.

Remember that the AutoCAD ActiveX Automation Help file is available—just press F1. If you are having trouble with a particular object, method, or property, highlight the object, method, or property in the VBA IDE and press F1.

In this chapter

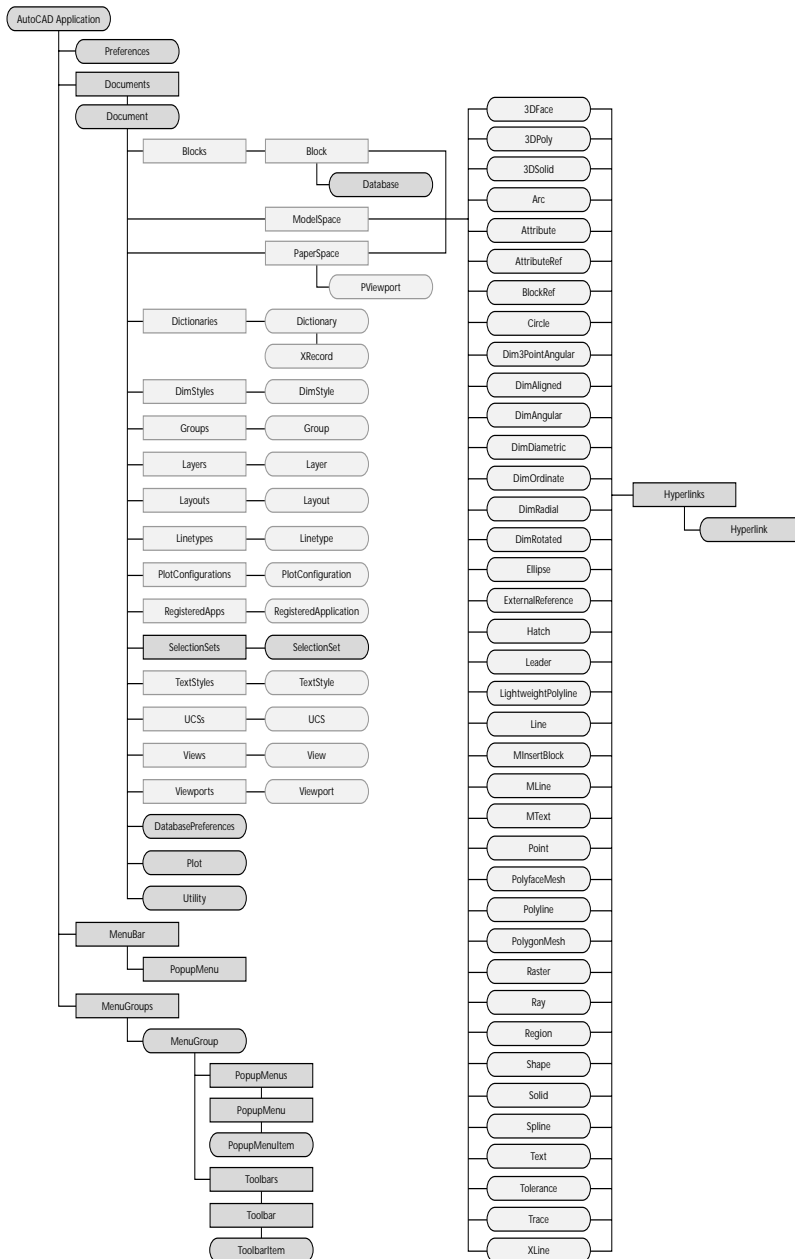
- Understanding the AutoCAD Object Model
- Accessing the Object Hierarchy
- Working with the Collection Objects
- Understanding Properties and Methods
- Understanding Parent Objects
- Locating the Type Library
- Using Variants in Methods and Properties
- Using Other Programming Languages

Understanding the AutoCAD Object Model

An object is the main building block of the AutoCAD ActiveX interface. Each exposed object represents a precise part of AutoCAD. There are many different types of objects in the AutoCAD ActiveX interface. For example

- Graphical objects such as lines, arcs, text, and dimensions are objects.
- Style settings such as linetypes and dimension styles are objects.
- Organizational structures such as layers, groups, and blocks are objects.
- The drawing display such as view and viewport are objects.
- Even the drawing and the AutoCAD application are considered objects.

The objects are structured in a hierarchical fashion, with the Application object at the root. The view of this hierarchical structure is referred to as the Object Model. The Object Model shows you which object provides access to the next level of objects.

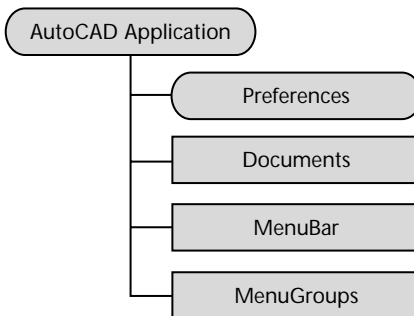


A Brief Look at the Application Object

The Application object is the Root object for the AutoCAD ActiveX Automation Object Model. From the Application object, you can access any of the other objects, or the properties or methods assigned to any object.

For example, the Application object has a Preferences property that returns the Preferences object. This object provides access to the registry-stored settings in the Options dialog box. (Drawing-stored settings are contained in the DatabasePreferences object, which will be discussed later.) Other properties of the Application object give you access to application-specific data such as the application name and version, and the AutoCAD size, location, and visibility. The methods of the Application object perform application-specific actions such as listing, loading, and unloading ADS and ARX applications, and quitting AutoCAD.

The Application object also provides links to the AutoCAD drawings through the Documents collection, the AutoCAD menus and toolbars through the MenuBar and MenuGroups collections, and the VBA IDE through a property called VBE.

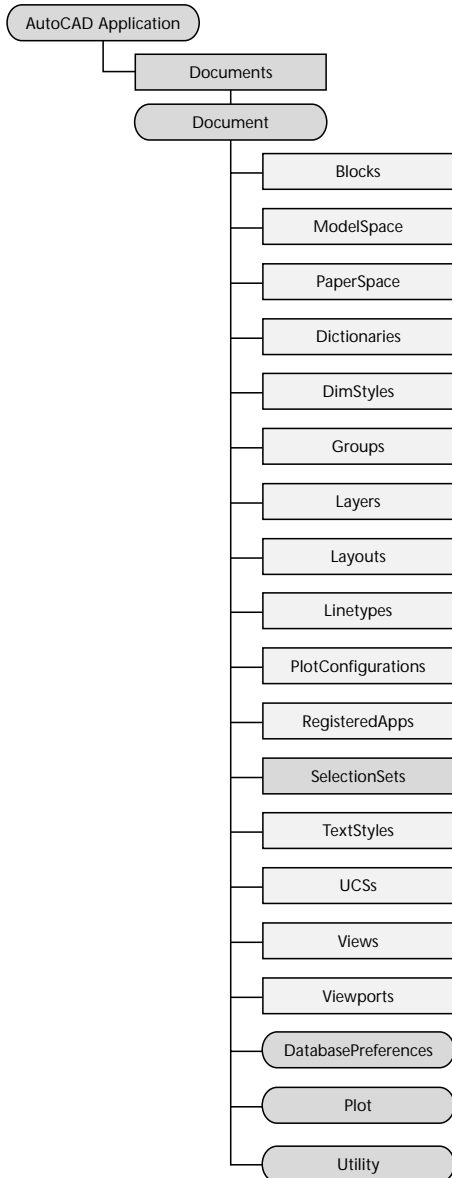


The application object is also the Global object for the ActiveX interface. This means that all the methods and properties for the Application object are available in the global name space.

A Brief Look at the Document Object

The Document object, which is actually an AutoCAD drawing, is found in the Documents collection and provides access to all of the graphical and most of the nongraphical AutoCAD objects. Access to the graphical objects (lines, circles, arcs, and so forth) is provided through the ModelSpace and PaperSpace collections, and access to nongraphical objects (layers, linetypes, text styles, and so forth) is provided through like-named collections such as

Layers, Linetypes, and TextStyles. The Document object also provides access to the Plot and Utility objects.



A Brief Look at the Collection Objects

AutoCAD groups most objects in collections. Although these collections contain different types of data, they can be processed using similar techniques. Each collection has a method for adding an object to the collection. Most collections use the Add method for this purpose. However, entity objects are usually added using a method titled Add<Entityname>. For example, to add a line you would use the AddLine method.

Collections also have some other methods and properties in common. The Count property can be used to obtain a zero-based count of the objects in a collection. The Item method can be used to obtain any object within a collection.

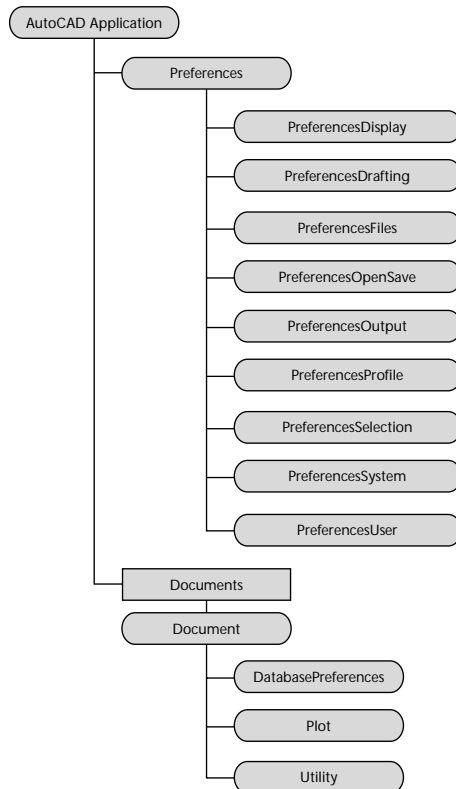
A Brief Look at the Graphical and Nongraphical Objects

Graphical objects, also known as entities, are the visible objects (lines, circles, raster images, and so forth) that make up a drawing. To create these objects, use the appropriate Add<Entityname> method. To modify or query these objects, use the methods or properties of the object itself. Each graphical object has methods that allow an application to perform most of the AutoCAD editing commands such as Copy, Erase, Move, Mirror, and so forth. These objects also have methods for setting and retrieving extended data (xdata), highlighting and updating, and retrieving the bounding box of the object. Graphical objects have typical properties such as Layer, Linetype, Color, and Handle. They also have specific properties, depending on their object type, such as Center, Radius, and Area.

Nongraphical objects are the invisible (informational) objects that are part of a drawing, such as Layers, Linetypes, DimStyles, SelectionSets, and so forth. To create these objects, use the Add method of the parent Collection object. To modify or query these objects, use the methods or properties of the object itself. Each nongraphical object has methods and properties specific to its purpose; all have methods for setting and retrieving extended data (xdata), and deleting themselves.

A Brief Look at the Preferences, Plot, and Utility Objects

Under the Preferences object is a set of objects, each corresponding to a tab in the Options dialog box. Together, these objects provide access to all the registry-stored settings in the Options dialog box. Drawing-stored settings are contained in the DatabasePreferences object. You can also set and modify options (and system variables that are not part of the Options dialog box) with the SetVariable and GetVariable methods. For more information on setting options see “Setting AutoCAD Preferences” on page 55.



The Plot object provides access to settings in the Plot dialog box and gives an application the ability to plot the drawing using various methods. For more information on plotting, see “Plotting Your Drawing” on page 276.

The Utility object provides user input and conversion functions. The user input functions are methods that prompt the user on the AutoCAD command line for input of various types of data, such as strings, integers, reals, points, and so forth. The conversion functions are methods that operate on AutoCAD-specific data types such as points and angles, in addition to string and number handling. For more information on the user input functions, see “Prompting for User Input” on page 75.

Accessing the Object Hierarchy

Accessing the object hierarchy is easy from within VBA. This is because VBA is running in-process with the current AutoCAD session so there is no additional step needed to connect it to the application.

VBA provides a link to the active drawing in the current AutoCAD session through the `ThisDrawing` object. By using `ThisDrawing` you gain immediate access to the current Document object and all of its methods and properties, and all of the other objects in the hierarchy.

When used in global projects, `ThisDrawing` always refers to the active document in AutoCAD. When used in embedded projects, `ThisDrawing` always refers to the document containing the project. For example, the following line of code in a global project saves whatever drawing is currently active in AutoCAD:

```
ThisDrawing.Save
```

Referencing Objects in the Object Hierarchy

You can reference objects directly or through a user-defined variable. To reference the objects directly, include the object in the calling hierarchy. For example, the following statement adds a line to the model space. Notice that the hierarchy starts with `ThisDrawing`, goes to the `ModelSpace` object, and then calls the `AddLine` method:

```
Dim startPoint(0 To 2) As Double, endPoint(0 To 2) As Double
Dim LineObj as AcadLine
startPoint(0) = 0: startPoint(1) = 0: startPoint(2) = 0
endPoint(0) = 30: endPoint(1) = 20: endPoint(2) = 0
Set LineObj = ThisDrawing.ModelSpace.AddLine(startPoint, endPoint)
```

To reference the objects through a user-defined variable, define the variable as desired type, then set the variable to the appropriate object. For example, the following code defines a variable (moSpace) of type AcadModel Space and sets the variable equal to the current model space:

```
Dim moSpace As AcadModel Space
Set moSpace = ThisDrawing.Model Space
```

The following statement then adds a line to the model space using the user-defined variable:

```
Dim startPoint(0 To 2) As Double, endPoint(0 To 2) As Double
Dim LineObj as AcadLine
startPoint(0) = 0: startPoint(1) = 0: startPoint(2) = 0
endPoint(0) = 30: endPoint(1) = 20: endPoint(2) = 0
Set LineObj = moSpace.AddLine(startPoint, endPoint)
```

Retrieving the first entity in model space

The following example returns the first entity object in model space. Similar code can do the same for paper space entities:

```
Sub Ch2_FindFirstEntity()
' This example returns the first entity in model space
On Error Resume Next

Dim entity As AcadEntity
If ThisDrawing.Model Space.count <> 0 Then
Set entity = ThisDrawing.Model Space.Item(0)
MsgBox entity.ObjectName + _
" is the first entity in model space."
Else
MsgBox "There are no objects in model space."
End If
End Sub
```

Accessing the Application Object

Because the ThisDrawing object provides a link to the Document object, you may be wondering how you access the root object (Application object), which is situated above the Document object in the object hierarchy. The Document object has a property called Application that provides a link to the Application object.

For example, the following line of code updates the application:

```
ThisDrawing.Application.Update
```

Working with the Collection Objects

A Collection object is a predefined object that contains (is a parent object for) all instances of a similar object. There is a Collection object for the following objects:

Documents Collection	Contains all documents open in the current AutoCAD session.
ModelSpace Collection	Contains all graphical objects (entities) in model space.
PaperSpace Collection	Contains all graphical objects (entities) in the active paper space layout.
Block Object	Contains all entities within a specific block definition.
Blocks Collection	Contains all blocks in the drawing.
Dictionaries Collection	Contains all dictionaries in the drawing.
DimStyles Collection	Contains all dimension styles in the drawing.
Groups Collection	Contains all groups in the drawing.
Hyperlinks Collection	Contains all hyperlinks for a given entity.
Layers Collection	Contains all layers in the drawing.
Layouts Collection	Contains all layouts in the drawing.
Linetypes Collection	Contains all linetypes in the drawing.
MenuBar Collection	Contains all menus currently displayed in AutoCAD.
MenuGroups Collection	Contains all menus and toolbars currently loaded in AutoCAD.
RegisteredApplications Collection	Contains all registered applications in the drawing.
SelectionSets Collection	Contains all selection sets in the drawing.
TextStyles Collection	Contains all text styles in the drawing.
UCSs Collection	Contains all user coordinates systems (UCS) in the drawing.
Views Collection	Contains all views in the drawing.
Viewports Collection	Contains all viewports in the drawing.

Accessing a Collection

Most collection objects are accessed through the Document object. The Document object contains a property for each of the Collection objects. For example, the following code defines a variable and sets it to the Layers collection of the current drawing:

```
Dim LayerCollection as AcadLayers  
Set LayerCollection = ThisDrawing.Layers
```

The Documents collection, MenuBar collection, and MenuGroups collection are accessed through the Application object. The Application object contains a property for each of these collections. For example, the following code defines a variable and sets it to the MenuGroups collection for the application:

```
Dim MenuGroupsCollection as AcadMenuGroups  
Set MenuGroupsCollection = ThisDrawing.Application.MenuGroups
```

Adding a New Member to a Collection Object

To add a new member to the collection, use the Add method. For example, the following code creates a new layer and adds it to the Layers collection:

```
Dim newLayer as AcadLayer  
Set newLayer = ThisDrawing.Layers.Add("MyNewLayer")
```

Iterating through a Collection Object

To select a specific member of a Collection object, use the Item method. The Item method requires an identifier as either an index number specifying the location of the item within the collection or a string representing the name of the item.

The Item method is the default method for a collection. If you do not specify a method name when referring to a collection, Item is assumed. The following statements are equivalent:

```
ThisDrawing.Layers.Item("ABC")  
ThisDrawing.Layers("ABC")
```

NOTE Do not use the entity edit methods (Copy, Array, Mirror, and so forth) on any object while simultaneously iterating through a collection using the For Each mechanism. Either finish your iteration before you attempt to edit an object in the collection or create a temporary array and set it equal to the collection. Then you can iterate through the copied array and perform your edits.

Iterating through the Layers collection

The following example iterates through a collection and displays the names of all layers in the collection:

```
Sub Ch2_IterateLayer()  
    ' Iterate through the collection  
    On Error Resume Next  
  
    Dim I As Integer  
    Dim msg As String  
    msg = ""  
    For I = 0 To ThisDrawing.Layers.Count - 1  
        msg = msg + ThisDrawing.Layers.Item(I).Name + vbCrLf  
    Next  
    MsgBox msg  
End Sub
```

Finding the layer named "ABC"

The following example refers to a layer named "ABC," and issues a message if the layer does not exist:

```
Sub Ch2_FindLayerABC()  
    ' Use the Item method to find a layer named "ABC"  
    On Error Resume Next  
  
    Dim ABCLayer As AcadLayer  
    Set ABCLayer = ThisDrawing.Layers("ABC")  
    If Err <> 0 Then  
        MsgBox "The layer 'ABC' does not exist."  
    End If  
End Sub
```

Deleting a Member of a Collection Object

To delete a specific dimension style, use the Delete method found on the member object. For example, the following code deletes the layer ABC:

```
Dim ABCLayer as AcadLayer  
Set ABCLayer = ThisDrawing.Layers.Item("ABC")  
ABCLayer.Delete
```

Once an object has been deleted, you must never attempt to access the object again later in the program.

Understanding Properties and Methods

Each object has associated properties and methods. Properties describe aspects of the individual object, while methods are actions that can be performed on the individual object. Once an object is created, you can query and edit the object through its properties and methods.

For example, a Circle object has the Center property. This property represents the 3D World Coordinate System coordinate at the center of that circle. To change the center of the circle, simply set this property to the new coordinate. The Circle object also has a method called Offset. This method creates a new object at a specified offset distance from the existing circle. To see a list of all properties and methods for the Circle object, refer to the Circle object in the AutoCAD *ActiveX and VBA Reference*.

Understanding Parent Objects

Each object has a parent object to which it is permanently linked. All objects originate from a single parent object called the root object. You can access all the objects in the interface by following the links from the root to the child objects. Additionally, all objects have a property called Application that links directly back to the root object.

The root object for the AutoCAD interface is the AutoCAD application.

Locating the Type Library

The objects, properties, and methods exposed by automation objects are contained in a type library. A type library is a file or part of a file that describes the type of one or more objects. Type libraries do not store objects; they store information. By accessing a type library, applications and browsers can determine the characteristics of an object, such as the interfaces supported by the object and the names and addresses of the members of each interface.

Before you can use the automation object exposed by an application, you must reference its type library. The reference is automatically set in AutoCAD VBA. For other interactive development environments you must create the reference.

You can use an application's objects without referencing the application's type library. However, it is preferable to add the type library reference for the following reasons:

- Globally accessible functions may be accessed directly without qualification.
- Invocation of functions, properties, and methods can be checked at compile time for correctness, and therefore will execute more quickly at runtime.
- It is possible to declare variables of the types defined in the library, which increases run-time reliability and readability.

Using Variants in Methods and Properties

AutoCAD ActiveX Automation uses variants to pass arrays of data. Although this may seem confusing to a novice user, it is not difficult once you learn the basics. In addition, AutoCAD ActiveX Automation provides utilities to help you convert your data types.

What Is a Variant?

A variant is a special data type that can contain any kind of data except fixed-length string data and user-defined types. A variant can also contain the special values `Empty`, `Error`, `Nothing`, and `NULL`. You can determine how the data in a variant is treated using the `VarType` or `TypeName` Visual Basic functions.

You can use the Variant data type in place of most any data type to work with data in a more flexible way.

Using Variants for Array Data

Variants are used to pass array data in and out of AutoCAD ActiveX Automation. This means that your arrays must be a variant to be accepted by AutoCAD ActiveX Automation methods and properties. In addition, array data output from AutoCAD ActiveX Automation must be handled as a variant.

NOTE In AutoCAD, VBA input arrays are automatically converted to variants. This means that you don't have to provide a variant array as input to the ActiveX Automation methods and properties when using them from VBA. However, all the output arrays will be in the form of variants, so remember to handle them appropriately.

Converting Arrays to Variants

AutoCAD ActiveX Automation provides a utility method to convert an array of data into a variant. This method is the `CreateTypedArray` method, which creates a variant that contains an array of integers, floating numbers, doubles, and so forth. You can pass the resulting variant into any AutoCAD method or property that accepts an array of numbers as a variant.

The `CreateTypedArray` method takes as input the type of values that are in the array, and the array of data to be converted. It returns the array of values as a variant.

Creating a spline using the `CreateTypedArray` method

The following code converts three arrays using `CreateTypedArray`: the coordinates for a spline's fit points, and the start and end tangent of the spline. It then passes the variant into the `AddSpline` method to create the spline.

```
Sub Ch2_CreateSplineUsingTypedArray()  
    ' This example creates a spline object in model space  
    ' using the CreateTypedArray method.  
    Dim splineObj As AcadSpline  
    Dim startTan As Variant  
    Dim endTan As Variant  
    Dim fitPoints As Variant  
  
    Dim utilObj As Object ' late bind the Utility object  
    Set utilObj = ThisDrawing.Utility  
  
    ' Define the Spline Object  
    utilObj.CreateTypedArray _  
        startTan, vbDouble, 0.5, 0.5, 0  
    utilObj.CreateTypedArray _  
        endTan, vbDouble, 0.5, 0.5, 0  
    utilObj.CreateTypedArray _  
        fitPoints, vbDouble, 0, 0, 0, 5, 5, 0, 10, 0, 0  
  
    Set splineObj = ThisDrawing.ModelSpace.AddSpline _  
        (fitPoints, startTan, endTan)  
  
    ' Zoom in on the newly created spline  
    ZoomAll  
End Sub
```

Interpreting Variant Arrays

Array information passed back from AutoCAD ActiveX Automation is passed back as a variant. If you know the data type of the array, you can simply access the variant as an array. If you don't know the data type contained in the variant, use the VBA functions `VarType` or `TypeOf`. These functions return the type of data in the variant. If you need to iterate through the array, you can use the VBA `For Each` statement.

Calculating the distance between two points

The following code demonstrates calculating the distance between two points input by the user. In this example, the data type is known because all coordinates are doubles. 3D coordinates are a three-element array of doubles and 2D coordinates are a two-element array of doubles.

```
Sub Ch2_CalculateDistance()  
    Dim point1 As Variant  
    Dim point2 As Variant  
  
    ' Get the points from the user  
    point1 = ThisDrawing.Utility.GetPoint _  
        (, vbCrLf & "First point: ")  
    point2 = ThisDrawing.Utility.GetPoint _  
        (point1, vbCrLf & "Second point: ")  
  
    ' Calculate the distance between point1 and point2  
    Dim x As Double, y As Double, z As Double  
    Dim dist As Double  
    x = point1(0) - point2(0)  
    y = point1(1) - point2(1)  
    z = point1(2) - point2(2)  
    dist = Sqr((Sqr((x ^ 2) + (y ^ 2)) ^ 2) + (z ^ 2))  
  
    ' Display the resulting distance  
    MsgBox "The distance between the points is: " _  
        & dist, , "Calculate Distance"  
End Sub
```

Using Other Programming Languages

This manual is written for the VBA programming language. The programming examples and sample applications are written in VBA. To use the code in other programming environments, you must update it for the chosen environment.

Use the documentation for your development environment to help you convert the example code.

Converting the VBA Code to VB

To update the coding examples for use with VB, you must first reference the AutoCAD type library. To do this in VB, select the References option from the Project menu to launch the Reference dialog box. From the References dialog box, choose AutoCAD Type Library, and then press OK.

Next, in the code example replace all references to `ThisDrawing` with a user-specified variable referencing the active document. To do this, define a variable for the AutoCAD application (`acadApp`) and for the current document (`acadDoc`). Then, set the application variable to the current AutoCAD application.

If AutoCAD is running, the VB `GetObject` function retrieves the AutoCAD Application object. If AutoCAD is not running, an error occurs that (in this example) is trapped, then cleared. The `CreateObject` function then attempts to create an AutoCAD Application object. If it succeeds, AutoCAD is started; if it fails, a message box displays a description of the error.

NOTE When running multiple sessions of AutoCAD, the `GetObject` function will return the first instance of AutoCAD in the Windows Running Object Table. See the Microsoft Visual Basic documentation on the Running Object Table (ROT) and the `GetObject` function for more information on verifying the session returned by `GetObject`.

You must set the AutoCAD application's `Visible` property to `TRUE` in order to display the AutoCAD drawing window.

NOTE If `GetObject` creates a new instance of AutoCAD (that is, AutoCAD was not already running when you issued `GetObject`), failure to set `Visible` to `TRUE` results in an invisible AutoCAD application; AutoCAD will not even appear on the Windows taskbar.

Connecting to AutoCAD from Visual Basic

The following code example uses the `Clear` and `Description` properties of `Err`. If your coding environment does not support these properties, you will need to modify the example appropriately:

```

Sub Ch2_ConnectToAcad()
    Dim acadApp As AcadApplication
    On Error Resume Next

    Set acadApp = GetObject(, "AutoCAD.Application")
    If Err Then
        Err.Clear
        Set acadApp = CreateObject("AutoCAD.Application")
        If Err Then
            MsgBox Err.Description
            Exit Sub
        End If
    End If
    MsgBox "Now running " + acadApp.Name + _
        " version " + acadApp.Version
End Sub

```

Next, set the document variable to the Document object in the AutoCAD application. The Document object is returned by the ActiveDocument property of the Application object.

```

Dim acadDoc as AcadDocument
Set acadDoc = acadApp.ActiveDocument

```

From this point on, use the acadDoc variable to reference the current AutoCAD drawing.

VBA versus VB Comparison Code Example

The following code example demonstrates creating a line in both VBA and VB.

Creating a line using VBA:

```
Sub Ch2_AddLineVBA()  
    ' This example adds a line  
    ' in model space  
    Dim lineObj As AcadLine  
    Dim startPoint(0 To 2) As Double  
    Dim endPoint(0 To 2) As Double  
  
    ' Define the start and end  
    ' points for the line  
    startPoint(0) = 1  
    startPoint(1) = 1  
    startPoint(2) = 0  
    endPoint(0) = 5  
    endPoint(1) = 5  
    endPoint(2) = 0  
  
    ' Create the line in model space  
    Set lineObj = ThisDrawing. _  
        ModelSpace.AddLine _  
        (startPoint, endPoint)  
  
    ' Zoom in on the newly created line  
    ZoomAll  
End Sub
```

Creating a line using VB:

```
Sub Ch2_AddLineVB()  
    On Error Resume Next  
  
    ' Connect to the AutoCAD application  
    Dim acadApp As AcadApplication  
    Set acadApp = GetObject _  
        (, "AutoCAD.Application")  
    If Err Then  
        Err.Clear  
        Set acadApp = CreateObject _  
            ("AutoCAD.Application")  
        If Err Then  
            MsgBox Err.Description  
            Exit Sub  
        End If  
    End If  
  
    ' Connect to the AutoCAD drawing  
    Dim acadDoc As AcadDocument  
    Set acadDoc = acadApp.ActiveDocument  
  
    ' Establish the endpoints of the line  
    Dim lineObj As AcadLine  
    Dim startPoint(0 To 2) As Double  
    Dim endPoint(0 To 2) As Double  
    startPoint(0) = 1  
    startPoint(1) = 1  
    startPoint(2) = 0  
    endPoint(0) = 5  
    endPoint(1) = 5  
    endPoint(2) = 0  
    ' Create a Line object in model space  
    Set lineObj = acadDoc.ModelSpace.AddLine _  
        (startPoint, endPoint)  
    ZoomAll  
    acadApp.Visible = True  
End Sub
```


Controlling the AutoCAD Environment

3

This chapter describes the fundamentals you need to know when developing an application in AutoCAD. It explains how to control the AutoCAD environment and how to work effectively in that environment.

In this chapter

- Opening, Saving, and Closing Drawings
- Setting AutoCAD Preferences
- Controlling the Application Window
- Controlling the Drawing Windows
- Resetting Active Objects
- Setting and Returning System Variables
- Drawing with Precision
- Prompting for User Input
- Accessing the AutoCAD Command Line
- Working with No Documents Open
- Importing Other File Formats
- Exporting to Other File Formats

Opening, Saving, and Closing Drawings

The Documents collection and Document object provide access to the AutoCAD file functions.

To create a new drawing, or open an existing drawing, use the methods on the Documents collection. The Add method creates a new drawing and adds that drawing to the Documents collection. The Open method opens an existing drawing. There is also a Close method on the Documents collection that closes all the drawings open in the AutoCAD session.

Use either the Save or SaveAs methods to save a drawing. Occasionally you will want to check if the active drawing has any unsaved changes. It is a good idea to do this before you quit the AutoCAD session or start a new drawing. Use the Saved property to make sure that the current drawing does not contain any unsaved changes.

To import and export drawings, use the Import and Export methods on the Document object.

Opening an existing drawing

This example uses the Open method to open an existing drawing. The Visual Basic Dir function is used to check for the existence of the file before trying to open it. You should change the drawing file name or path to specify an existing AutoCAD drawing file on your system.

```
Sub Ch3_OpenDrawing()  
    Dim dwgName As String  
    dwgName = "c:\Program Files\autocad 2002\sample\campus.dwg"  
    If Dir(dwgName) <> "" Then  
        ThisDrawing.Application.Documents.Open dwgName  
    Else  
        MsgBox "File " & dwgName & " does not exist."  
    End If  
End Sub
```

Creating a new drawing

This example uses the Add method to create a new drawing based on the default template.

```
Sub Ch3_NewDrawing()  
    Dim docObj As AcadDocument  
    Set docObj = ThisDrawing.Application.Documents.Add  
End Sub
```

Saving the active drawing

The following example saves the active drawing under its current name and again under a new name.

```
Sub Ch3_SaveActiveDrawing()  
    ' Save the active drawing under the current name  
    ThisDrawing.Save  
  
    ' Save the active drawing under a new name  
    ThisDrawing.SaveAs "MyDrawing.dwg"  
End Sub
```

Testing if a drawing has unsaved changes

This example checks to see if there are unsaved changes and verifies with the user that it is OK to save the drawing (if it is not OK, skip to the end). If OK, use the Save method to save the current drawing, as shown here:

```
Sub Ch3_TestIfSaved()  
    If Not (ThisDrawing.Saved) Then  
        If MsgBox("Do you wish to save this drawing?", _  
            vbYesNo) = vbYes Then  
            ThisDrawing.Save  
        End If  
    End If  
End Sub
```

Setting AutoCAD Preferences

There are nine objects pertaining to options, each representing a tab on the Options dialog box. These objects provide access to all of the registry-stored options in the Options dialog box. You can customize many of the AutoCAD settings by using properties found on these objects. These objects are

- PreferencesDisplay
- PreferencesDrafting
- PreferencesFiles
- PreferencesOpenSave
- PreferencesOutput
- PreferencesProfiles
- PreferencesSelection
- PreferencesSystem
- PreferencesUser

These objects are accessible via the Preferences object. To gain access to the Preferences object, use the Preferences property of the Application object:

```
Dim acadPref as AcadPreferences
Set acadPref = ThisDrawing.Application.Preferences
```

You can then access any of the specific Preferences objects using the Display, Drafting, Files, OpenSave, Output, Profile, Selection, System, and User properties.

Setting the crosshairs to full screen

```
Sub Ch2_PrefsSetCursor()
    ' This example sets the crosshairs of the AutoCAD drawing cursor
    ' to full screen.

    ' Access the Preferences object
    Dim acadPref As AcadPreferences
    Set acadPref = ThisDrawing.Application.Preferences

    ' Use the CursorSize property to set the size of the crosshairs
    acadPref.Display.CursorSize = 100
End Sub
```

Displaying the screen menu and scroll bars

```
Sub Ch2_PrefsSetDisplay()
    ' This example enables the screen menu and disables the scroll
    ' bars with the DisplayScreenMenu and DisplayScrollBars
    ' properties.

    ' Access the Preferences object
    Dim acadPref As AcadPreferences
    Set acadPref = ThisDrawing.Application.Preferences

    ' Display the screen menu and disable scroll bars
    acadPref.Display.DisplayScreenMenu = True
    acadPref.Display.DisplayScrollBars = False
End Sub
```

Database Preferences

In addition to the nine Preferences objects, the DatabasePreferences object contains all the options stored in the drawing. This separate object was provided to make the drawing-stored options available to applications accessing AutoCAD drawings without first starting the AutoCAD application (ObjectDBX™ applications).

The DatabasePreferences object is found under the Document object.

Controlling the Application Window

The ability to control the Application window allows developers the flexibility to create effective and intelligent applications. There will be times when it is appropriate for your application to minimize the AutoCAD window, perhaps while your code is performing work in another application such as Excel. Additionally, you will often want to verify the state of the AutoCAD window before performing such tasks as prompting for input from the user.

Using methods and properties found on the Application object, you can change the position, size, and visibility of the Application window. You can also use the WindowState property to minimize, maximize, and check the current state of the Application window.

Positioning and sizing the Application window

This example uses the WindowTop, WindowLeft, Width, and Height properties to position the AutoCAD Application window in the upper-left corner of the screen and size it to 400 pixels wide by 400 pixels high.

```
Sub Ch3_PositionApplicationWindow()  
    ThisDrawing.Application.WindowTop = 0  
    ThisDrawing.Application.WindowLeft = 0  
    ThisDrawing.Application.Width = 400  
    ThisDrawing.Application.Height = 400  
End Sub
```

Maximizing the Application window

```
Sub Ch3_MaximizeApplicationWindow()  
    ThisDrawing.Application.WindowState = acMax  
End Sub
```

Minimizing the Application window

```
Sub Ch3_MinimizeApplicationWindow()  
    ThisDrawing.Application.WindowState = acMin  
End Sub
```

Finding the current state of the Application window

This example queries the state of the Application window and displays the state in a message box to the user.

```

Sub Ch3_CurrentWindowState()
    Dim CurrWindowState As Integer
    Dim msg As String
    CurrWindowState = ThisDrawing.Application.WindowState
    msg = Choose(CurrWindowState, "normal", _
        "minimized", "maximized")
    MsgBox "The application window is " + msg
End Sub

```

Making the Application window invisible

The following code uses the Visible property to make the AutoCAD application invisible to the end user.

```

Sub Ch3_HideWindowState()
    ThisDrawing.Application.Visible = False
End Sub

```

Controlling the Drawing Windows

Like the AutoCAD Application window, you can minimize, maximize, reposition, resize, and check the state of any Document window. But you can also change the way the drawing is displayed within a window by using views, viewports, and zooming methods.

AutoCAD ActiveX provides many ways to display views of your drawing. You can control the drawing display to move quickly to different areas of your drawing while you track the overall effect of your changes. You can zoom to change magnification or pan to reposition the view in the graphics area, save a view and then restore it when you need to plot or refer to specific details, or display several views at one time by splitting the screen into several tiled viewports.

Positioning and Sizing the Document Window

Use the Document object to modify the position and size of any document window. The Document window can be minimized or maximized by using the WindowState property, and you can find the current state of the Document window by using the WindowState property.

Positioning a Document window

This example uses the Width and Height properties to set the active document window to 400 pixels wide by 400 pixels high.

```
Sub Ch3_Si zeDocumentWi ndow()
ThisDrawing.Width = 400
ThisDrawing.Height = 400
End Sub
```

Maximizing the active Document window

```
Sub Ch3_Maxi mi zeDocumentWi ndow()
ThisDrawing.WindowState = acMax
End Sub
```

Minimizing the active Document window

```
Sub Ch3_Mi ni mi zeDocumentWi ndow()
ThisDrawing.WindowState = acMi n
End Sub
```

Finding the current state of the active document window

```
Sub Ch3_CurrentWi ndowState()
Dim CurrWi ndowState As Integer
Dim msg As String
CurrWi ndowState = ThisDrawing.WindowState
msg = Choose(CurrWi ndowState, "normal", _
             "mi ni mi zed", "maxi mi zed")
MsgBox "The document window is " + msg
End Sub
```

Using Zoom

A view is a specific magnification, position, and orientation of a drawing. The most common way to change a view is to use one of the many AutoCAD Zoom options, which increases or decreases the size of the image displayed in the graphics area. For more information on zooming in AutoCAD, see “Magnify a View (Zoom)” in the *User’s Guide*.

Defining a Zoom Window

You can quickly zoom in on an area by specifying the corners that define it. To zoom in on an area by specifying its boundaries, use either the `ZoomWindow` or `ZoomPickWindow` method. The `ZoomWindow` method allows you to define two points representing the Zoom window programmatically. The `ZoomPickWindow` requires the user to pick two points. These two picked points become the Zoom window.

Zooming the active drawing to a window defined by two points

```
Sub Ch3_ZoomWindow()  
    ' ZoomWindow  
    MsgBox "Perform a ZoomWindow with: " & vbCrLf & _  
        "1.3, 7.8, 0" & vbCrLf & _  
        "13.7, -2.6, 0", , "ZoomWindow"  
  
    Dim point1(0 To 2) As Double  
    Dim point2(0 To 2) As Double  
    point1(0) = 1.3: point1(1) = 7.8: point1(2) = 0  
    point2(0) = 13.7: point2(1) = -2.6: point2(2) = 0  
    ThisDrawing.Application.ZoomWindow point1, point2  
    ' ZoomPickWindow  
    MsgBox "Perform a ZoomPickWindow", , "ZoomPickWindow"  
  
    ThisDrawing.Application.ZoomPickWindow  
End Sub
```

Scaling a View

If you need to increase or decrease the magnification of the image by a precise scale, you can specify a zoom scale in three ways:

- Relative to the drawing limits
- Relative to the current view
- Relative to paper space units

To scale a view, use the `ZoomScaled` method. This method takes two parameters as input: the scale and the type of scale. The scale is simply a number. How that number gets interpreted by AutoCAD depends on the type of scale you choose.

The type of scale determines if the scale value is created relative to the drawing limits, the current view, or the paper space units. To scale relative to the drawing limits, use the constant `acZoomScaledAbsolute`. To scale the view relative to the current view, use the constant `acZoomScaledRelative`. To scale relative to paper space units, use the constant `acZoomScaledRelativePSpace`.

Zooming the active drawing using a specified scale

```
Sub Ch3_ZoomScaled()  
    MsgBox "Perform a ZoomScaled using: " & vbCrLf & _  
        "Scale Type: acZoomScaledRelative" & vbCrLf & _  
        "Scale Factor: 2", , "ZoomScaled"  
  
    Dim scaleFactor As Double  
    Dim scaleType As Integer  
  
    scaleFactor = 2  
    scaleType = acZoomScaledRelative  
  
    ThisDrawing.Application.ZoomScaled scaleFactor, scaleType  
End Sub
```

Centering Objects

You can move a specific point in your drawing to the center of the graphics area. The **ZoomCenter** method is useful for resizing an object and bringing it to the center of the viewport. With **ZoomCenter**, you can specify a scale size by entering a magnification relative to the current view.

Zooming the active drawing to a specified center

The following example shows the effects of using **ZoomCenter** to display a view at the same size and at twice the size:

```
Sub Ch3_ZoomCenter()  
    MsgBox "Perform a ZoomCenter using: " & vbCrLf & _  
        "Center 3, 3, 0" & vbCrLf & _  
        "Magnification: 10", , "ZoomCenter"  
  
    Dim Center(0 To 2) As Double  
    Dim magnification As Double  
  
    Center(0) = 3: Center(1) = 3: Center(2) = 0  
    magnification = 10  
  
    ThisDrawing.Application.ZoomCenter Center, magnification  
End Sub
```

Displaying Drawing Limits and Extents

To display a view based on the drawing boundaries or the extents of the objects in the drawing, use the **ZoomAll**, **ZoomExtents**, or **ZoomPrevious** methods.

ZoomAll displays the entire drawing. If the objects extend beyond the limits, **ZoomAll** displays the extents of the objects. If the objects are drawn within the limits, **ZoomAll** displays the limits.

ZoomExtents calculates zooms based on the extents of the active viewport, not the current view. Usually the active viewport is entirely visible, so the results are obvious and intuitive. However, when using the Zoom methods in model space while working in a paper space viewport, if you are zoomed in beyond the paper space viewport's borders, some of the area zoomed may not be visible.

ZoomExtents changes the view to encompass the entity extents for the current drawing. In some cases (for both ZoomAll and ZoomExtents), this may cause a regeneration. Regeneration will not occur on layers that are frozen or turned off. If your drawing has no objects, ZoomExtents displays the drawing limits.

For 3D views, ZoomAll and ZoomExtents have the same effect. Infinite construction lines (xlines) and rays do not affect either option.

ZoomPrevious zooms the current viewport to its previous extents.

See “Magnify a View (Zoom)” in the *User's Guide* for illustrations of how zooming works.

Zooming the active drawing to all contents and to the drawing extents

```
Sub Ch3_ZoomAll ()  
  ZoomAll  
  MsgBox "Perform a ZoomAll", , "ZoomAll"  
  ThisDrawing.Application.ZoomAll  
  
  ZoomExtents  
  MsgBox "Perform a ZoomExtents", , "ZoomExtents"  
  ThisDrawing.Application.ZoomExtents  
End Sub
```

Using Named Views

You can name and save a view you want to reuse. When you no longer need the view, you can delete it.

To create a new view, use the Add method to add a new view to the Views collection. When you save the drawing, the viewing position and scale of the view are saved.

You name the view when you create it. The name of the view can be up to 255 characters long and contain letters, digits, and the special characters dollar sign (\$), hyphen (-), and underscore (_).

To delete a named view, simply use the Delete method. The Delete method for the View object lies on the View object, not its parent.

Adding a new View object

```
Sub Ch3_AddView()  
    ' Add a named view to the views collection  
    Dim viewObj As AcadView  
    Set viewObj = ThisDrawing.Views.Add("View1")  
End Sub
```

Deleting a view object

The following example deletes a View object (viewObj).

```
Sub Ch3_DeleteView()  
    Dim viewObj As AcadView  
    Set viewObj = ThisDrawing.Views("View1")  
    ' Delete the view  
    viewObj.Delete  
End Sub
```

Deleting a named view from the Views collection

This example deletes a named view from the Views collection.

```
Sub Ch3_DeleteViewFromCollection()  
    ThisDrawing.Views("View1").Delete  
End Sub
```

Using Tiled Viewports

AutoCAD usually begins a new drawing using a single viewport that fills the entire graphics area. You can split the graphics area to display several viewports simultaneously. For example, if you keep both the full and the detail views visible, you can see the effects of your detail changes on the entire drawing. In each tiled viewport, you can do the following:

- Zoom, set the Snap, Grid, and UCS icon modes, and restore named views in individual viewports
- Draw from one viewport to another when executing a command
- Name a configuration of viewports so you can reuse it

You can display tiled viewports in various configurations. How you display the viewports depends on the number and size of the views you need to see.

For further information and illustrations describing viewports, see “Set Model Tab Viewports” in the *User's Guide*.

Splitting the Active Viewport

To split the active viewport, use the `Split` method. This method takes one parameter, the type of configuration to split the viewport into. To specify the window configuration, use one of the following constants that correspond to the default configurations previously shown: `acViewport2Horizontal`, `acViewport2Vertical`, `acViewport3Left`, `acViewport3Right`, `acViewport3Horizontal`, `acViewport3Vertical`, `acViewport3Above`, `acViewport3Below`, or `acViewport4`.

For further information on changing viewport configuration, see “Set Model Tab Viewports” in the *User’s Guide*.

Splitting a viewport into two horizontal windows

The following example creates a new viewport and then splits the viewport into two horizontal windows.

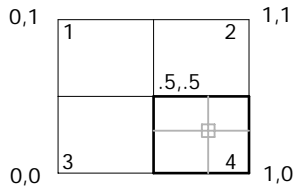
```
Sub SplitAViewport()  
    ' Create a new viewport  
    Dim vportObj As AcadViewport  
    Set vportObj = ThisDrawing.Viewports.Add("TEST_VIEWPORT")  
  
    ' Split vportObj into 2 horizontal windows  
    vportObj.Split acViewport2Horizontal  
  
    ' Now set vportObj to be the active viewport  
    ThisDrawing.ActiveViewport = vportObj  
End Sub
```

Making Another Tiled Viewport Current

You enter points and select objects in the current viewport. To make a viewport current, use the `ActiveViewport` property.

You can iterate through existing viewports to find a particular viewport. To do this, first identify the name of the viewport configuration on which the desired viewport resides using the `Name` property. Additionally, if the viewport configuration has been split, each individual viewport on the configuration can be identified through the `LowerLeftCorner` and `UpperRightCorner` properties.

The `LowerLeftCorner` and `UpperRightCorner` properties represent the graphic placement of the viewport on the display. These properties are defined as follows (using a four-way split as an example):



In this example:

- Viewport 1-LowerLeftCorner = (0, .5), UpperRightCorner = (.5, 1)
- Viewport 2-LowerLeftCorner = (.5, .5), UpperRightCorner = (1, 1)
- Viewport 3-LowerLeftCorner = (0, 0), UpperRightCorner = (.5, .5)
- Viewport 4-LowerLeftCorner = (.5, 0), UpperRightCorner = (1, .5)

Splitting a viewport, then iterating through the windows

This example splits a viewport into four windows. It then iterates through all the viewports in the drawing and displays the viewport name and the lower-left and upper-right corner for each viewport.

```
Sub Ch3_IteratingViewportWindows()
    ' Create a new viewport and make it active
    Dim vportObj As AcadViewport
    Set vportObj = ThisDrawing.Viewports.Add("TEST_VIEWPORT")
    ThisDrawing.ActiveViewport = vportObj

    ' Split vport into 4 windows
    vportObj.Split acViewport4

    ' Iterate through the viewports,
    ' highlighting each viewport and displaying
    ' the upper right and lower left corners
    ' for each.
    Dim vport As AcadViewport
    Dim LLCorner As Variant
    Dim URCorner As Variant
    For Each vport In ThisDrawing.Viewports
        ThisDrawing.ActiveViewport = vport
        LLCorner = vport.LowerLeftCorner
        URCorner = vport.UpperRightCorner
        MsgBox "Viewport: " & vport.Name & " is now active." & _
            vbCrLf & "Lower left corner: " & _
            LLCorner(0) & ", " & LLCorner(1) & vbCrLf & _
            "Upper right corner: " & _
            URCorner(0) & ", " & URCorner(1)
    Next vport
End Sub
```

Updating the Geometry in the Document Window

Many of the actions you perform through AutoCAD ActiveX Automation modify what is displayed in the AutoCAD drawing. Not all of these actions immediately update the display of the drawing. This is designed so you can make several changes to the drawing without waiting for the display to update after every single action. Instead, you can bundle your actions together and make a single call to update the display when you have finished.

The methods that will update the display are `Update` and `Regen`.

The `Update` method updates the display of a single object only. The `Regen` method regenerates the entire drawing and recomputes the screen coordinates and view resolution for all objects. It also reindexes the drawing database for optimum display and object selection performance.

Updating the display of a single object

This example creates a circle and then colors the circle red. It then updates the circle using the `Update` method so that the circle is visible in AutoCAD.

```
Sub Ch3_UpdateDisplay()  
    Dim circleObj As AcadCircle  
    Dim center(0 To 2) As Double  
    Dim radius As Double  
    center(0) = 1: center(1) = 1: center(2) = 0  
    radius = 1  
  
    ' Create the circle and then color it red  
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)  
    circleObj.Color = acRed  
  
    ' Update the circle  
    circleObj.Update  
End Sub
```

Resetting Active Objects

Changes to most active objects, such as the active layer and active linetype, will appear immediately. However, there are several active objects that must be reset for changes to appear. These objects are the active text style, the active UCS, and the active viewport. If changes are made to any of these objects, the object must be reset, and the `Regen` method must be called for the changes to appear.

To reset these objects, simply set the `ActiveTextStyle`, `ActiveUCS`, or `ActiveViewport` property, using the updated object.

Resetting the active viewport

The following example changes the display of the grid in the active viewport and then resets the viewport as the active viewport to display the change.

```
Sub Ch3_ResetActiveViewport()  
    ' Toggle the setting of the grid display  
    ' for the active viewport  
    ThisDrawing.ActiveViewport.GridOn = _  
        Not (ThisDrawing.ActiveViewport.GridOn)  
  
    ' Reset the active viewport  
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport  
End Sub
```

Setting and Returning System Variables

The `Document` object provides the `SetVariable` and `GetVariable` methods for setting and retrieving AutoCAD system variables. For example, to assign an integer to the `MAXSORT` system variable, use the following code:

```
ThisDrawing.SetVariable "MAXSORT", 100
```

Drawing with Precision

With AutoCAD you can create your drawings with precise geometry without performing tedious calculations. Often you can specify precise points without knowing the coordinates. Without leaving the drawing screen, you can perform calculations on your drawing and display various types of status information.

At this time, AutoCAD ActiveX Automation does not provide a method for the following AutoCAD capabilities:

- Setting object snaps
- Specifying measured intervals on objects or dividing objects into segments

For more information about this topic, see chapter 15, “Use Precision Tools,” in the *User's Guide*.

Adjusting Snap and Grid Alignment

You can use the grid as a visual guideline and turn on Snap mode to restrict cursor movement. In addition to setting the spacing, you can adjust the snap and grid alignment. You can rotate the alignment, or you can set it for use with isometric drawings.

If you need to draw along a specific alignment or angle, you can rotate the snap angle. The center point of the snap angle rotation is the snap base point. If you need to align a hatch pattern, you can change this point, which is normally set to 0,0.

To rotate the snap angle, use the `SnapRotationAngle` property. To change the base point of the snap angle rotation, use the `SnapBasePoint` property.

NOTE Both properties require a call to the `Update` method to update the AutoCAD display.

See “Adjust Grid and Grid Snap” in the *User’s Guide* for more information on using and setting snaps and grids.

Changing the snap base point and rotation angle

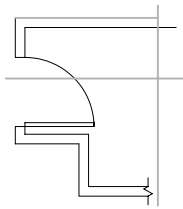
This example changes the snap base point to (1,1) and the snap rotation angle to 30 degrees. The grid is turned on so that the changes are visible.

```
Sub Ch3_ChangeSnapBasePoint()  
    ' Turn on the grid for the active viewport  
    ThisDrawing.ActiveViewport.GridOn = True  
  
    ' Change the snap base point to 1, 1  
    Dim newBasePoint(0 To 1) As Double  
    newBasePoint(0) = 1: newBasePoint(1) = 1  
    ThisDrawing.ActiveViewport.SnapBasePoint = newBasePoint  
  
    ' Change the snap rotation angle to 30 degrees (0.575 radians)  
    Dim rotationAngle As Double  
    rotationAngle = 0.575  
    ThisDrawing.ActiveViewport.SnapRotationAngle = rotationAngle  
  
    ' reset the viewport  
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport  
End Sub
```

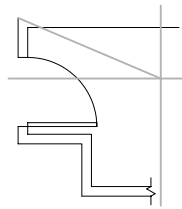

Using Ortho Mode

As you draw lines or move objects, you can use Ortho mode to restrict the cursor to the horizontal or vertical axis. (The orthogonal alignment depends on the current snap angle or UCS.) Ortho mode works with activities that require you to specify a second point. You can use Ortho not only to establish vertical or horizontal alignment but also to enforce parallelism or create regular offsets.

By allowing AutoCAD to impose orthogonal restraints, you can draw more quickly. For example, you can create a series of perpendicular lines by turning on Ortho mode before you start drawing. Because the lines are constrained to the horizontal and vertical axes, you can draw faster, knowing that the lines are perpendicular.



ortho mode on



ortho mode off

As you move the cursor, a rubber-band line that defines the displacement follows the horizontal or vertical axis, depending on which axis is nearest to the cursor. AutoCAD ignores Ortho mode in perspective views, or when you enter coordinates on the command line or specify an object snap.

To turn Ortho mode on or off, use the `OrthoOn` property. This property requires a Boolean for input. Set to `TRUE` to turn Ortho mode on, and to `FALSE` to turn Ortho mode off. For example, the following statement turns Ortho mode on for the active viewport:

```
ThisDrawing.ActiveViewport.OrthoOn = True
```

Drawing Construction Lines

You can create construction lines that extend to infinity in one or both directions. Construction lines that extend in one direction are known as rays. Construction lines that extend in both directions are known as xlines. These construction lines can be used as a reference for creating other objects. For example, you can use construction lines to find the center of a triangle, prepare multiple views of the same item, or create temporary intersections that you can use for object snaps.

For more information about construction lines, see “Draw Construction and Reference Geometry” in chapter 16, “Draw Geometric Objects,” in the *User’s Guide*.

Creating Construction XLines

A construction xline can be placed anywhere in 3D space and extends to infinity in both directions. To create an xline, use the `AddXLine` method. This method specifies the line by the two-point method, you enter or select two points to define the orientation. The first point, the root, is considered the midpoint of the construction line.

For more information on creating construction xlines, see “Draw Construction and Reference Geometry” in the *User’s Guide*.

Adding a construction line

The following example code creates an `XLine` object using the two points (5, 0, 0) and (1, 1, 0).

```
Sub Ch3_AddXLine()  
    Dim xlineObj As AcadXLine  
    Dim basePoint(0 To 2) As Double  
    Dim directionVec(0 To 2) As Double  
  
    ' Define the xline  
    basePoint(0) = 2#: basePoint(1) = 2#: basePoint(2) = 0#  
    directionVec(0) = 1#: directionVec(1) = 1#: directionVec(2) = 0#  
  
    ' Create the xline in model space  
    Set xlineObj = ThisDrawing.ModelSpace.AddXLine _  
        (basePoint, directionVec)  
    ThisDrawing.Application.ZoomAll  
End Sub
```

Querying Construction XLines

Once created, you can query the first point of a construction xline using the `BasePoint` property. The second point used to create the xline is not stored with the object. Instead, use the `DirectionVector` property to obtain the directional vector for the xline.

Querying a construction line

This example finds the base point and directional vector for the xline created in “Adding a construction line.”

```
Dim BPoint As Variant
Dim Vector As Variant

BPoint = xlineObj.BasePoint
Vector = xlineObj.DirectionVector
```

Creating Rays

A ray is a line in 3D space that starts at a point you specify and extends to infinity. Unlike xline construction lines, which extend in two directions, rays extend in only one direction. As a result, rays help reduce the visual clutter caused by numerous construction lines.

Like construction lines, rays are ignored by commands that display the drawing extents.

For more information on rays, see “Draw Construction Lines (and Rays)” in the *User’s Guide*.

Querying Rays

Once created, you can query the first point of a ray using the BasePoint property. The second point used to create the ray is not stored with the object. Instead, use the DirectionVector property to obtain the directional vector for the ray.

Adding, querying, and editing a Ray object

The following example code creates a Ray object using the two points (5, 0, 0) and (1, 1, 0). It then queries the current base point and direction vector and displays the results in a message box. The direction vector is then changed and the base point and new direction vector are queried and displayed.

```

Sub Ch3_EditRay()
    Dim rayObj As AcadRay
    Dim basePoint(0 To 2) As Double
    Dim secondPoint(0 To 2) As Double

    ' Define the ray
    basePoint(0) = 3#: basePoint(1) = 3#: basePoint(2) = 0#
    secondPoint(0) = 4#: secondPoint(1) = 4#: secondPoint(2) = 0#

    ' Creates a Ray object in model space
    Set rayObj = ThisDrawing.ModelSpace.AddRay _
        (basePoint, secondPoint)
    ThisDrawing.Application.ZoomAll

    ' Find the current status of the Ray
    MsgBox "The base point of the ray is: " & _
        rayObj.basePoint(0) & ", " & _
        rayObj.basePoint(1) & ", " & _
        rayObj.basePoint(2) & vbCrLf & _
        "The directional vector for the ray is: " & _
        rayObj.DirectionVector(0) & ", " & _
        rayObj.DirectionVector(1) & ", " & _
        rayObj.DirectionVector(2), , "Edit Ray"

    ' Change the directional vector for the ray
    Dim newVector(0 To 2) As Double
    newVector(0) = -1
    newVector(1) = 1
    newVector(2) = 0
    rayObj.DirectionVector = newVector
    ThisDrawing.Regen False
    MsgBox "The base point of the ray is: " & _
        rayObj.basePoint(0) & ", " & _
        rayObj.basePoint(1) & ", " & _
        rayObj.basePoint(2) & vbCrLf & _
        "The directional vector for the ray is: " & _
        rayObj.DirectionVector(0) & ", " & _
        rayObj.DirectionVector(1) & ", " & _
        rayObj.DirectionVector(2), , "Edit Ray"
End Sub

```

Calculating Points and Values

By using the methods provided by the Utility object, you can quickly solve a mathematical problem or locate points on your drawing. By using methods on the Utility object, you can do the following:

- Find the angle of a line from the *X* axis with the *AngleFromXAxis* method
- Convert an angle as a string to a real (double) value with the *AngleToReal* method
- Convert an angle from a real (double) value to a string with the *AngleToString* method

- Convert a distance from a string to a real (double) value with the `DistanceToReal` method
- Create a variant that contains an array of integers, floating numbers, doubles, and so forth, with the `CreateTypedArray` method
- Find the point at a specified angle and distance from a given point with the `PolarPoint` method
- Translate a point from one coordinate system to another coordinate system with the `TranslateCoordinates` method
- Find the distance between two points entered by the user with the `GetDistance` method

Finding the distance between two points using the `GetDistance` method

This example uses the `GetDistance` method to obtain the point coordinates, and the `MsgBox` function to display the calculated distance.

```
Sub Ch3_GetDistanceBetweenTwoPoints()
    Dim returnDist As Double

    ' Return the value entered by user. A prompt is provided.
    returnDist = ThisDrawing.Utility.GetDistance _
        (, "Pick two points.")
    MsgBox "The distance between the two points is: " & returnDist
End Sub
```

Calculating Areas

You can find the area of an arc, circle, ellipse, lightweight polyline, polyline, region, or planar-closed spline by using the `Area` property.

If you need to calculate the combined area of more than one object, you can keep a running total as you add or use the `Boolean` method on a series of regions to obtain a single region representing the desired area. From this single region you can use the `Area` property to obtain its area.

The calculated area differs according to the type of object you query. For an explanation of how area is calculated for each object type, see “Obtain Area Information” in chapter 15, “Use Precision Tools,” in the *User's Guide*.

Calculating a Defined Area

You can measure an arbitrary closed region defined by the 2D or 3D points specified by the user. The points must be coplanar.

To obtain the area specified by points from the user

- 1 Use the `GetPoint` method in a loop to obtain the points from the user.
- 2 Create a lightweight polyline from the points provided by the user. Use the `AddLightweightPolyline` method to create the polyline.
- 3 Use the `Area` property to obtain the area of the newly created polyline.
- 4 Erase the polyline using the `Erase` method.

Calculating the area defined by points entered from the user

This example prompts the user to enter five points. A polyline is then created out of the points entered. The polyline is closed, and the area of the polyline is displayed in a message box.

```
Sub Ch3_CalculateDefinedArea()  
    Dim p1 As Variant  
    Dim p2 As Variant  
    Dim p3 As Variant  
    Dim p4 As Variant  
    Dim p5 As Variant  
  
    ' Get the points from the user  
    p1 = ThisDrawing.Utility.GetPoint(, vbCrLf & "First point: ")  
    p2 = ThisDrawing.Utility.GetPoint(p1, vbCrLf & "Second point: ")  
    p3 = ThisDrawing.Utility.GetPoint(p2, vbCrLf & "Third point: ")  
    p4 = ThisDrawing.Utility.GetPoint(p3, vbCrLf & "Fourth point: ")  
    p5 = ThisDrawing.Utility.GetPoint(p4, vbCrLf & "Fifth point: ")  
  
    ' Create the 2D polyline from the points  
    Dim polyObj As AcadLWPolyline  
    Dim vertices(0 To 9) As Double  
    vertices(0) = p1(0): vertices(1) = p1(1)  
    vertices(2) = p2(0): vertices(3) = p2(1)  
    vertices(4) = p3(0): vertices(5) = p3(1)  
    vertices(6) = p4(0): vertices(7) = p4(1)  
    vertices(8) = p5(0): vertices(9) = p5(1)  
    Set polyObj = ThisDrawing.ModelSpace.AddLightweightPolyline _  
        (vertices)  
    polyObj.Closed = True  
    ThisDrawing.Application.ZoomAll  
  
    ' Display the area for the polyline  
    MsgBox "The area defined by the points is " & _  
        polyObj.Area, , "Calculate Defined Area"  
End Sub
```

Prompting for User Input

The Utility object, which is a child of the Document object, defines the user input methods. The user input methods display a prompt on the AutoCAD command line and request input of various types. This type of user input is most useful for interactive input of screen coordinates, entity selection, and short-string or numeric values. If your application requires the input of numerous options or values, a dialog box may be more appropriate than individual prompts.

Each user input method displays a prompt on the AutoCAD command line and returns a value specific to the type of input requested. For example, GetString returns a string, GetPoint returns a variant (which contains a three-element array of doubles), and GetInteger returns an integer value. You can further control the input from the user with the InitializeUserInput method. This method lets you control things such as NULL input (pressing ENTER), input of zero or negative numbers, and input of arbitrary text values.

To force the prompt to display on a line by itself, use the carriage return/line-feed constant (vbCrLf) at the beginning of your prompt strings.

GetString Method

The GetString method prompts the user for input of a string at the AutoCAD command prompt. This method accepts two parameters. The first parameter controls the input of spaces in the input string. If it is set to 0, spaces are not allowed (SPACEBAR terminates the input); if set to 1, the string can contain spaces (ENTER must be used to terminate the input). The second parameter is the prompt string.

Getting a string value from the user at the AutoCAD command line

The following example displays the Enter Your Name prompt, and requires that the input from the user be terminated by pressing ENTER (spaces are allowed in the input string). The string value is stored in the retVal variable and is displayed using a message box.

```
Sub Ch3_GetStringFromUser()  
    Dim retVal As String  
    retVal = ThisDrawing.Utility.GetString _  
        (1, vbCrLf & "Enter your name: ")  
    MsgBox "The name entered was: " & retVal  
End Sub
```

The GetString method does not honor a prior call to the InitializeUserInput method.

GetPoint Method

The GetPoint method prompts the user for the specification of a point at the AutoCAD command prompt. This method accepts two parameters, an optional from point and the prompt string. If the from point is provided, AutoCAD draws a rubber-band line from that point. To control the user input, this method can be preceded by a call to the InitializeUserInput method.

Getting a point selected by the user

The following example prompts the user for two points, then draws a line using those points as the start point and endpoint.

```
Sub Ch3_GetPointsFromUser()  
    Dim startPnt As Variant  
    Dim endPnt As Variant  
    Dim prompt1 As String  
    Dim prompt2 As String  
  
    prompt1 = vbCrLf & "Enter the start point of the line: "  
    prompt2 = vbCrLf & "Enter the end point of the line: "  
  
    ' Get the first point without entering a base point  
    startPnt = ThisDrawing.Utility.GetPoint(, prompt1)  
  
    ' Use the point entered above as the base point  
    endPnt = ThisDrawing.Utility.GetPoint(startPnt, prompt2)  
  
    ' Create a line using the two points entered  
    ThisDrawing.ModelSpace.AddLine startPnt, endPnt  
    ThisDrawing.Application.ZoomAll  
End Sub
```

GetKeyword Method

The GetKeyword method prompts the user for input of a keyword at the AutoCAD command prompt. This method accepts only one parameter, which is the prompt string. The keywords and input parameters are defined with a call to the InitializeUserInput method.

Getting a keyword from the user at the AutoCAD command line

The following example forces the user to enter a keyword by setting the first parameter of InitializeUserInput to 1, which disallows NULL input (pressing ENTER). The second parameter establishes the list of valid keywords.


```

Sub Ch3_KeyWord()
    Dim keyWord As String
    ThisDrawing.Utility.InitializeUserInput 1, "Line Circle Arc"
    keyWord = ThisDrawing.Utility.GetKeyword _
        (vbCrLf & "Enter an option (Line/Circle/Arc): ")
    MsgBox keyWord, , "GetKeyword Example"
End Sub

```

A more user-friendly keyword prompt is one that provides a default value if the user presses ENTER (NULL input). Notice the minor modifications to the following example:

```

Sub Ch3_KeyWord2()
    Dim keyWord As String
    ThisDrawing.Utility.InitializeUserInput 0, "Line Circle Arc"
    keyWord = ThisDrawing.Utility.GetKeyword _
        (vbCrLf & "Enter an option (Line/Circle/<Arc>): ")
    If keyWord = "" Then keyWord = "Arc"
    MsgBox keyWord, , "GetKeyword Example"
End Sub

```

Controlling User Input

You can use the `InitializeUserInput` method to define keywords or restrict the type of input to the user input method. The use and parameter values are similar to the `AutoLISP` `initget` function. `InitializeUserInput` can be used with the following methods: `GetAngle`, `GetCorner`, `GetDistance`, `GetInteger`, `GetKeyword`, `GetOrientation`, `GetPoint`, and `GetReal`. `InitializeUserInput` cannot be used with the `GetString` method. Use the `GetInput` method to retrieve the string value (keyword or arbitrary input) when the user input method does not return a string value.

The `InitializeUserInput` method accepts two parameters. The first parameter is a bit-coded integer value that determines the input options for the user input method. The second parameter is a string that defines the valid keywords.

Getting an integer value or a keyword from the user at the AutoCAD command line

The following example prompts the user for a positive, non-negative integer value or a keyword:

```

Sub Ch3_UserInput()

    ' The first parameter of InitializeUserInput (6)
    ' restricts input to positive and non-negative
    ' values. The second parameter is the list of
    ' valid keywords.
    ThisDrawing.Utility.InitializeUserInput 6, "Big Small Regular"

    ' Set the prompt string variable
    Dim promptStr As String
    promptStr = vbCrLf & "Enter the size or (Big/Small/<Regular>): "

    ' At the GetInteger prompt, entering a keyword or pressing
    ' ENTER without entering a value results in an error. To allow
    ' your application to continue and check for the error
    ' description, you must set the error handler to resume on error.
    On Error Resume Next

    ' Get the value entered by the user
    Dim returnInteger As Integer
    returnInteger = ThisDrawing.Utility.GetInteger(promptStr)

    ' Check for an error. If the error number matches the
    ' one shown below, then use GetInput to get the returned
    ' string; otherwise, use the value of returnInteger.

    If Err.Number = -2145320928 Then
        Dim returnString As String
        Debug.Print Err.Description
        returnString = ThisDrawing.Utility.GetInput()
        If returnString = "" Then ' ENTER returns null string
            returnString = "Regular" ' Set to default
        End If
        Err.Clear
    Else
        returnString = returnInteger ' Otherwise,
        ' Use the value entered
    End If

    ' Display the result
    MsgBox returnString, , "InitializeUserInput Example"

End Sub

```

Accessing the AutoCAD Command Line

You can send commands directly to the AutoCAD command line by using the `SendCommand` method. The `SendCommand` method sends a single string to the command line. The string must contain the arguments to the command listed in the order expected by the prompt sequence of the executed command. A blank space or the ASCII equivalent of a carriage return in the string is equivalent to pressing ENTER on the keyboard. Unlike the

AutoLISP environment, invoking the SendCommand method with no argument is invalid.

Sending a command to the AutoCAD command line

The following example creates a circle with a center of (2, 2, 0) and a radius of 4. The drawing is then zoomed to all the geometry in the drawing. Notice that there is a space at the end of the string which represents the final ENTER to begin execution of the command.

```
Sub Ch3_SendACommandToAutoCAD()  
  ThisDrawing.SendCommand "_Circle 2,2,0 4 "  
  ThisDrawing.SendCommand "_zoom a "  
End Sub
```

Working with No Documents Open

AutoCAD always starts up with a new or existing document open. It is possible, however, to close all documents during the current session.

If you close all the documents in the AutoCAD user interface, you will notice a few changes to the application window. The available menus are reduced to simply the File, View, Window, and Help menus. These menus also have reduced available options on them. You will also notice that there is no command line.

Similarly, the ActiveX interface only allows the following actions when no documents are open:

- You can open a document.
- You can create a new document.
- You can import a document.
- You can exit out of AutoCAD.

These actions are all available from the Documents collection. The methods and properties of the Documents collection, in addition to a limited set of methods and properties of the Application object, are the only valid interface available when there are no documents open. If you perform any other action, such as attempting to access user options, your actions will result in an error.

Use the Count property on the Documents collection to determine if AutoCAD is in a zero document state. If Documents.Count = 0, then AutoCAD is in a zero document state. If Documents.Count > 0, then there is at least one drawing open.

It is also important to note that in VBA the `ThisDrawing` object is not defined when AutoCAD is in a zero document state. This makes sense since `ThisDrawing` normally refers to the active drawing and in the zero document state there are no drawings open. Attempting to execute a macro that uses `ThisDrawing` will result in a run-time error. To avoid the error, use the VBA `GetObject` function to obtain a connection to AutoCAD when there are no documents open.

Importing Other File Formats

You can use drawings or images from other applications by opening them in specific formats. AutoCAD handles some form of conversion for DXF, SAT, and WMF files. For all versions, you can import the file by using the `Import` method. This method takes three values as input: the name of the file to import, the insertion point in the drawing to place the file, and the scale factor to use when placing the imported drawing.

Exporting to Other File Formats

If you need to use an AutoCAD drawing in another application, you can convert it to a specific format by using the `Export` method. This method exports the AutoCAD drawing to a WMF, SAT, EPS, DXF, or BMP format. The `Export` method takes three values as input: the name for the new file to be created, the extension for the new file, and the selection set of objects to export.

When exporting to WMF, SAT, or BMP formats, you must provide a non-empty selection set. This selection set specifies the objects from the drawing to export. If no selection set is specified, nothing is exported and a trappable invalid argument error results.

When exporting to EPS and DXF formats, `Export` ignores the selection set argument, but it is still required. The entire drawing is automatically exported for these formats.

Exporting a drawing as a DXF file and importing it again

This example creates a circle in the current drawing. It then exports the drawing to a file called *DXFExprt.DXF*, opens a new drawing, and imports the file. Note that an empty selection set is provided as an argument to Export. The Export method ignores selection set information when exporting a DXF file, but a syntax error results if the argument is omitted.

```
Sub Ch3_ImportingAndExporting()

    ' Create the circle for visual representation
    Dim circleObj As AcadCircle
    Dim centerPt(0 To 2) As Double
    Dim radius As Double
    centerPt(0) = 2: centerPt(1) = 2: centerPt(2) = 0
    radius = 1
    Set circleObj = ThisDrawing.ModelSpace.AddCircle _
        (centerPt, radius)
    ThisDrawing.Application.ZoomAll

    ' Create an empty selection set
    Dim sset As AcadSelectionSet
    Set sset = ThisDrawing.SelectionSets.Add("NEWSSET")

    ' Export the current drawing to a DXF file in the
    ' AutoCAD temporary file directory
    Dim tempPath As String
    Dim exportFile As String
    Const dxfname As String = "DXFExprt"
    tempPath = _
        ThisDrawing.Application.preferences.Files.TempFilePath
    exportFile = tempPath & dxfname
    ThisDrawing.Export exportFile, "DXF", sset

    ' Delete the empty selection set
    ThisDrawing.SelectionSets.Item("NEWSSET").Delete

    ' Open a new drawing
    ThisDrawing.Application.Documents.Add "acad.dwt"

    ' Define the import
    Dim importFile As String
    Dim insertPoint(0 To 2) As Double
    Dim scaleFactor As Double
    importFile = tempPath & dxfname & ".dxf"
    insertPoint(0) = 0: insertPoint(1) = 0: insertPoint(2) = 0
    scaleFactor = 2#

    ' Import the file
    ThisDrawing.Import importFile, insertPoint, scaleFactor
    ThisDrawing.Application.ZoomAll

End Sub
```


Creating and Editing AutoCAD Entities

4

You can create a range of objects, from simple lines and circles to spline curves, ellipses, and associative hatch areas. In general, you add objects to model space using one of the Add methods. You can also create objects in paper space, or in a block.

Once an object is created, you can change the layer, color, and linetype of the object. You can also add text to annotate your drawing.

In this chapter

- Creating Objects
- Working with Selection Sets
- Editing Objects
- Using Layers, Colors, and Linetypes
- Saving and Restoring Layer Settings
- Adding Text to Drawings

Creating Objects

While there are often several different ways to create the same graphical object in AutoCAD, ActiveX Automation offers only one creation method per object. For example, in AutoCAD there are four different ways you can create a circle: (1) by specifying the center and radius, (2) by two points defining the diameter, (3) by three points defining the circumference, or (4) by two tangents and a radius. However, in ActiveX Automation there is only one creation method provided to create a circle, and that method uses the center and radius.

NOTE The VB and VBA methods of creating objects using either `CreateObject` or `Dim` with the `New` keyword can only be used to create the AutoCAD Application object. All other AutoCAD objects must be created using the `Add` or `Add<objectname>` methods provided in the AutoCAD interface.

Determining the Container Object

Graphical objects are created in either the `ModelSpace` collection, `PaperSpace` collection, or a `Block` object.

The `ModelSpace` collection is returned by the `ModelSpace` property and the `PaperSpace` collection by the `PaperSpace` property.

You can reference these objects directly, or through a user-defined variable. To reference the objects directly, include the object in the calling hierarchy. For example, the following statement adds a line to the model space:

```
Set LineObj = ThisDrawing.ModelSpace.AddLine(startPoint, endPoint)
```

To reference the objects through a user-defined variable, define the variable as type `AcadModelSpace` or `AcadPaperSpace`, and then set the variable to the appropriate property of the active document. The following example defines two variables and sets them equal to the current model space and paper space, respectively:

```
Dim moSpace As AcadModelSpace
Dim paSpace As AcadPaperSpace
Set moSpace = ThisDrawing.ModelSpace
Set paSpace = ThisDrawing.PaperSpace
```

The following statement adds a line to the model space using the user-defined variable:

```
Set LineObj = moSpace.AddLine(startPoint, endPoint)
```


Creating Lines

The line is the most basic object in AutoCAD. You can create a variety of lines—single lines, and multiple line segments with and without arcs. In general, you draw lines by specifying coordinate points. The default linetype is CONTINUOUS, an unbroken line, but various linetypes are available that use dots and dashes.

To create a line, use one of the following methods:

AddLine Creates a line passing through two points.

AddLightweightPolyline

Creates a 2D lightweight polyline from a list of vertices.

AddMLine Creates a multiline.

AddPolyline Creates a 2D or 3D polyline.

Standard lines and multilines are created on the *XY* plane of the WCS. Polylines and Lightweight Polylines are created in the Object Coordinate System (OCS). For information about converting OCS coordinates, see “Converting Coordinates” on page 249.

Creating a Polyline object

This example uses the **AddLightweightPolyline** method to create a simple two-segment polyline using the 2D coordinates (2,4), (4,2), and (6,4).

```
Sub Ch4_AddLightweightPolyline()  
    Dim plineObj As AcadLWPolyline  
    Dim points(0 To 5) As Double  
  
    ' Define the 2D polyline points  
    points(0) = 2: points(1) = 4  
    points(2) = 4: points(3) = 2  
    points(4) = 6: points(5) = 4  
  
    ' Create a lightweight Polyline object in model space  
    Set plineObj = ThisDrawing.ModelSpace.AddLightweightPolyline(points)  
    ThisDrawing.Application.ZoomAll  
End Sub
```

Creating Curved Objects

You can create a variety of curved objects with AutoCAD, including spline curves, circles, arcs, and ellipses. All curves are created on the *XY* plane of the current WCS.

To create a curve, use one of the following methods:

<code>AddArc</code>	Creates an arc given the center, radius, start and end angles.
<code>AddCircle</code>	Creates a circle given the center point and radius.
<code>AddEllipse</code>	Creates an ellipse given the center point, a point on the major axis, and the radius ratio.
<code>AddSpline</code>	Creates a quadratic or cubic NURBS (nonuniform rational B-spline) curve.

Creating a Spline object

This example creates a spline in model space using three points (0, 0, 0), (5, 5, 0), and (10, 0, 0). The spline has start and end tangents of (0.5, 0.5, 0.0).

```
Sub Ch4_CreateSpline()  
    ' This example creates a spline object in model space.  
    ' Declare the variables needed  
    Dim splineObj As AcadSpline  
    Dim startTan(0 To 2) As Double  
    Dim endTan(0 To 2) As Double  
    Dim fitPoints(0 To 8) As Double  
  
    ' Define the variables  
    startTan(0) = 0.5: startTan(1) = 0.5: startTan(2) = 0  
    endTan(0) = 0.5: endTan(1) = 0.5: endTan(2) = 0  
    fitPoints(0) = 1: fitPoints(1) = 1: fitPoints(2) = 0  
    fitPoints(3) = 5: fitPoints(4) = 5: fitPoints(5) = 0  
    fitPoints(6) = 10: fitPoints(7) = 0: fitPoints(8) = 0  
  
    ' Create the spline  
    Set splineObj = ThisDrawing.ModelSpace.AddSpline _  
        (fitPoints, startTan, endTan)  
    ZoomAll  
End Sub
```

For more information about splines, see the *Spline object* and *AddSpline* method documentation in the *AutoCAD ActiveX and VBA Reference*.

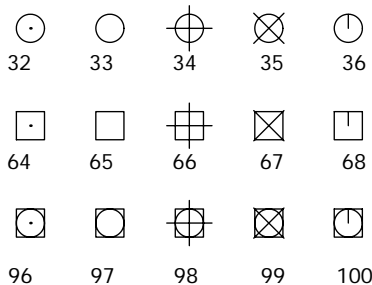
Creating Point Objects

Point objects can be useful, for example, as node or reference points that you can snap to and offset objects from. You can set the style of the point and its size relative to the screen or in absolute units.

The PDMODE and PDSIZE system variables control the appearance of Point objects. The PDMODE values 0, 2, 3, and 4 specify a figure to draw through the point. A value of 1 selects nothing to be displayed.



Adding 32, 64, or 96 to the previous value selects a shape to draw around the point in addition to the figure drawn through it:



PDSIZE controls the size of the point figures, except for PDMODE values 0 and 1. A 0 setting generates the point at 5 percent of the graphics area height. A positive PDSIZE value specifies an absolute size for the point figures. A negative value is interpreted as a percentage of the viewport size. The size of all points is recalculated when the drawing is regenerated.

After you change PDMODE and PDSIZE, the appearance of existing points changes the next time the drawing is regenerated.

To set PDMODE and PDSIZE, use the SetVariable method.

Creating a Point object and changing its appearance

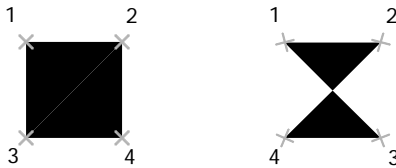
The following code example creates a Point object in model space at the coordinate (5, 5, 0). The PDMODE and PDSIZE system variables are then updated.

```
Sub Ch4_CreatePoint()  
  Dim pointObj As AcadPoint  
  Dim location(0 To 2) As Double  
  
  ' Define the location of the point  
  location(0) = 5#: location(1) = 5#: location(2) = 0#  
  
  ' Create the point  
  Set pointObj = ThisDrawing.ModelSpace.AddPoint(location)  
  ThisDrawing.SetVariable "PDMODE", 34  
  ThisDrawing.SetVariable "PDSIZE", 1  
  ZoomAll  
End Sub
```

Creating Solid-Filled Areas

You can create triangular and quadrilateral areas filled with a color. For quicker results, create these areas with the FILLMODE system variable off, then turn FILLMODE on to fill the finished area.

When you create a quadrilateral solid-filled area, the sequence of the third and fourth points determines its shape. Compare the following illustrations:



The first two points define one edge of the polygon. The third point is defined diagonally opposite from the second. If the fourth point is set equal to the third point, then a filled triangle is created.

To create a solid-filled area, use the AddSolid method.

For more information about filling solids, see “Create Solid-Filled Areas” in chapter 18, “Hatches and Fills,” in the *User's Guide*.

Creating a solid-filled object

The following code example creates a quadrilateral solid in model space using the coordinates (0, 0, 0), (5, 0, 0), (5, 8, 0), and (0, 8, 0).

```

Sub Ch4_CreateSolid()
    Dim solidObj As AcadSolid
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    Dim point3(0 To 2) As Double
    Dim point4(0 To 2) As Double

    ' Define the solid
    point1(0) = 0#: point1(1) = 0#: point1(2) = 0#
    point2(0) = 5#: point2(1) = 0#: point2(2) = 0#
    point3(0) = 5#: point3(1) = 8#: point3(2) = 0#
    point4(0) = 0#: point4(1) = 8#: point4(2) = 0#
    ' Create the solid object in model space
    Set solidObj = ThisDrawing.ModelSpace.AddSolid _
        (point1, point2, point3, point4)

    ZoomAll
End Sub

```

Working with Regions

Regions are two-dimensional enclosed areas you create from closed shapes called loops. A loop is a curve or a sequence of connected curves that defines an area on a plane with a boundary that does not intersect itself. Loops can be combinations of lines, lightweight polylines, circles, arcs, ellipses, elliptical arcs, splines, 3D faces, traces, and solids. The objects that make up the loops must either be closed or form closed areas by sharing endpoints with other objects. They must also be coplanar (on the same plane). The loops that make up a region must be defined as an array of objects.

For more information about working with regions, see “Create and Combine Areas (Regions)” in chapter 16, “Draw Geometric Objects,” in the *User’s Guide*.

Creating Regions

To create a region, use the `AddRegion` method. This method will create a region out of every closed loop formed by the input array of curves. AutoCAD converts closed 2D and planar 3D polylines to separate regions, then converts polylines, lines, and curves that form closed planar loops. If more than two curves share an endpoint, the resulting region might be arbitrary. Because of this, several regions may actually be created when using the `AddRegion` method. Use a variant to hold the newly created array of regions.

To calculate the total number of Region objects created, use the `UBound` and `LBound` Visual Basic functions, as in the following example:

```
UBound(objRegions) - LBound(objRegions) + 1
```

where `objRegions` is a variant containing the return value from `AddRegion`. This statement will calculate the total number of regions created.

Creating a simple region

The following code example creates a region from a single circle.

```
Sub Ch4_CreateRegion()  
    ' Define an array to hold the  
    ' boundaries of the region.  
    Dim curves(0 To 0) As AcadCircle  
  
    ' Create a circle to become a  
    ' boundary for the region.  
    Dim center(0 To 2) As Double  
    Dim radius As Double  
    center(0) = 2  
    center(1) = 2  
    center(2) = 0  
    radius = 5#  
    Set curves(0) = ThisDrawing.ModelSpace.AddCircle _  
        (center, radius)  
  
    ' Create the region  
    Dim regionObj As Variant  
    regionObj = ThisDrawing.ModelSpace.AddRegion(curves)  
  
    ZoomAll  
End Sub
```

Creating Composite Regions

You can create composite regions by subtracting, combining, or finding the intersection of regions or 3D solids. You can then extrude or revolve composite regions to create complex solids. To create a composite region, use the Boolean method.

When you subtract one region from another, you call the Boolean method from the first region. This is the region from which you want to subtract. For example, to calculate how much carpeting is needed for a floorplan, call the Boolean method from the outer boundary of the floor space and use the uncarpeted areas, such as pillars and counters, as the object in the Boolean parameter list.

Creating a composite region

```
Sub Ch4_CreateCompositeRegions()  
    ' Create two circles, one representing a room,  
    ' the other a pillar in the center of the room  
    Dim RoomObjects(0 To 1) As AcadCircle  
    Dim center(0 To 2) As Double  
    Dim radius As Double  
    center(0) = 4  
    center(1) = 4  
    center(2) = 0  
    radius = 2#  
    Set RoomObjects(0) = ThisDrawing.ModelSpace.AddCircle(center, radius)  
  
    radius = 1#  
    Set RoomObjects(1) = ThisDrawing.ModelSpace.AddCircle(center, radius)  
  
    ' Create a region from the two circles  
    Dim regions As Variant  
    regions = ThisDrawing.ModelSpace.AddRegion(RoomObjects)  
  
    ' Copy the regions into the region variables for ease of use  
    Dim RoundRoomObj As AcadRegion  
    Dim PillarObj As AcadRegion  
  
    If regions(0).Area > regions(1).Area Then  
        ' The first region is the room  
        Set RoundRoomObj = regions(0)  
        Set PillarObj = regions(1)  
    Else  
        ' The first region is the pillar  
        Set PillarObj = regions(0)  
        Set RoundRoomObj = regions(1)  
    End If  
  
    ' Color the room object red and the pillar object cyan  
    RoundRoomObj.Color = acRed  
    PillarObj.Color = acCyan  
    ZoomAll  
  
    ' Subtract the pillar space from the floor space to  
    ' get a region that represents the total carpet area.  
    RoundRoomObj.Boolean acSubtraction, PillarObj  
  
    ' Use the Area property to determine the total carpet area  
    MsgBox "The carpet area is: " & RoundRoomObj.Area  
End Sub
```

Find the area of the resulting region with the Area property.

Uniting Regions

To unite regions, call the Boolean method and enter the constant `acUnion` for the operation instead of `acSubtraction`. You can combine regions in any order to unite them.

Finding the Intersection of Two Regions

To find the intersection of two regions, use the constant `acIntersection`. You can combine regions in any order to intersect them.

Creating Hatches

Hatching fills a specified area in a drawing with a pattern.

When creating a hatch, you do not initially specify the area to be filled. First you must create the Hatch object. Once this is done, you can specify the outer loop, which is the outermost boundary for the hatch. You can then continue to specify any inner loops that may exist in the hatch.

For more information about working with hatches, see “Overview of Hatch Patterns and Solid Fills” in chapter 18, “Hatches and Fills,” in the *User’s Guide*.

Creating the Hatch Object

When creating the Hatch object, you specify the hatch pattern type, the hatch pattern name, and the associativity. Once the Hatch object has been created, you will not be able to change the hatch associativity.

To create a Hatch object, use the `AddHatch` method.

Associating a Hatch

You can create associative or nonassociative hatches. Associative hatches are linked to their boundaries and updated when the boundaries are modified. Nonassociative hatches are independent of their boundaries.

Associativity can only be set when a hatch is created. Once a hatch has been created, you can unassociate it, but you cannot associate it again.

To make a hatch associative, set the `Associativity` parameter of the `AddHatch` method to `TRUE`. To make a hatch nonassociative, set the `Associativity` parameter of the `AddHatch` method to `FALSE`.

Assigning the Hatch Pattern Type and Name

AutoCAD supplies a solid-fill and more than fifty industry-standard hatch patterns. Hatch patterns highlight a particular feature or area of a drawing. For example, patterns can help differentiate the components of a 3D object or represent the materials that make up an object.

You can use a pattern supplied with AutoCAD or one from an external pattern library. For a table of the hatch patterns supplied with AutoCAD, see appendix E, “Standard Libraries,” in the AutoCAD *Command Reference*.

To specify a unique pattern, you must enter both a pattern type and a pattern name when creating the Hatch object. The pattern type specifies where to look up the pattern name. When entering the pattern type, use one of the following constants:

`acHatchPatternTypePredefined`

Selects the pattern name from those defined in the `acad.pat` file

`acHatchPatternTypeUserDefined`

Defines a pattern of lines using the current linetype

`acHatchPatternTypeCustomDefined`

Selects the pattern name from a PAT other than the *acad.pat* file

When entering the pattern name, use a name that is valid for the file specified by the pattern type.

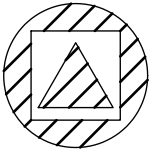
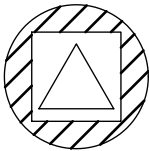
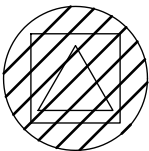
Defining the Hatch Boundaries

Once the Hatch object is created, the hatch boundaries can be added. Boundaries can be any combination of lines, arcs, circles, 2D polylines, ellipses, splines, and regions.

The first boundary added must be the outer boundary, which defines the outer most limits to be filled by the hatch. To add the outer boundary, use the `AppendOuterLoop` method.

Once the outer boundary is defined, you can continue adding inner boundaries. Add inner boundaries with the `AppendInnerLoop` method.

Inner boundaries define islands within the hatch. How these islands are handled by the Hatch object depends on the setting of the [HatchStyle](#) property. The HatchStyle property can be set to one of the following conditions:

Hatch style definitions		
HatchStyle	Condition	Description
	Normal	Specifies standard style, or normal. This option hatches inward from the outermost area boundary. If AutoCAD encounters an internal boundary, it turns off hatching until it encounters another boundary. This is the default setting for the HatchStyle property.
	Outer	Fills the outermost areas only. This style also hatches inward from the area boundary, but it turns off hatching if it encounters an internal boundary and does not turn it back on again.
	Ignore	Ignores internal structure. This option hatches through all internal objects.

When you have finished defining the hatch it must be evaluated before it can be displayed. Use the Evaluate method to do this.

Creating a Hatch object

This example creates an associate hatch in model space. Once the hatch has been created, you can change the size of the circle that the hatch is associated with. The hatch will change to match the current circle size.

```

Sub Ch4_CreateHatch()
    Dim hatchObj As AcadHatch
    Dim patternName As String
    Dim PatternType As Long
    Dim bAssociativity As Boolean

    ' Define the hatch
    patternName = "ANSI 31"
    PatternType = 0
    bAssociativity = True

    ' Create the associative Hatch object
    Set hatchObj = ThisDrawing.ModelSpace.AddHatch _
        (PatternType, patternName, bAssociativity)

    ' Create the outer boundary for the hatch. (a circle)
    Dim outerLoop(0 To 0) As AcadEntity
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 3: center(1) = 3: center(2) = 0
    radius = 1
    Set outerLoop(0) = ThisDrawing.ModelSpace.AddCircle(center, radius)

    ' Append the outerboundary to the hatch
    ' object, and display the hatch
    hatchObj.AppendOuterLoop (outerLoop)
    hatchObj.Evaluate
    ThisDrawing.Regen True

End Sub

```

Working with Selection Sets

A selection set is a group of AutoCAD objects specified for processing as a single unit. A selection set can consist of a single object, or it can be a more complex grouping: for example, the set of objects of a certain color on a certain layer.

Defining a selection set is a two-step process. First, you must create a new selection set and add it to the SelectionSets collection. Once created, you then populate the selection set with the objects you want to process.

Creating a Selection Set

To create a named selection set, use the Add method. This method requires only a single parameter—the name of the selection set.

If a selection set of the same name already exists, AutoCAD returns an error message. It is a good programming practice to delete a selection set when you no longer need it. Use the Delete method to delete a selection set, as in the following example:

```
ThisDrawing.SelectionSets.Item("NewSelectionSet").Delete
```

Creating an empty selection set

This example creates a new selection set.

```
Sub Ch4_CreateSelectionSet()  
    Dim selectionSet1 As AcadSelectionSet  
    Set selectionSet1 = ThisDrawing.SelectionSets._  
        Add("NewSelectionSet")  
End Sub
```

Adding Objects to a Selection Set

You can add objects to the active selection set by using any of the following methods:

AddItems	Adds one or more objects to the specified selection set.
Select	Selects objects and places them into the active selection set. You can select all objects, objects within and crossing a rectangular area, objects within and crossing a polygon area, all objects crossing a fence, the most recently created object, the objects in the most recent selection set, objects within a window, or objects within a window polygon.
SelectAtPoint	Selects objects passing through a given point and places them into the active selection set.
SelectByPolygon	Selects objects within a fence and adds them to the active selection set.
SelectOnScreen	Prompts the user to pick objects from the screen and adds them into the active selection set.

Adding selected objects to a selection set

This example prompts the user to select objects, then adds those objects to the selection set. Then it changes the color of the objects in the selection set to blue.

```
Sub Ch4_AddToASelctionSet()  
    ' Create a new selection set  
    Dim sset As AcadSelectionSet  
    Set sset = ThisDrawing.SelectionSets.Add("SS1")  
  
    ' Prompt the user to select objects  
    ' and add them to the selection set.  
    ' To finish selecting, press ENTER.  
    sset.SelectOnScreen  
  
    ' Iterate through the selection set  
    ' and color each object blue  
    Dim entry As AcadEntity  
    For Each entry In sset  
        entry.Color = acBlue  
        entry.Update  
    Next entry  
End Sub
```

Defining Rules for Selection Sets

You can limit selection sets by property, such as color, or by object type using filter lists. For example, you can copy only the blue objects in a circuit board drawing, or only objects on a certain layer. You can also combine selection criteria in your filter list. For example, you can tell AutoCAD to include an object in a selection set only if it is a blue circle on a specific layer. Filter lists can be specified for the `Select`, `SelectAtPoint`, `SelectByPolygon`, and `SelectOnScreen` methods.

NOTE Filtering recognizes only colors or linetypes explicitly assigned to objects, not those inherited by the layer.

Using Filter Lists to Define Selection Set Rules

Filter lists are composed of pairs of arguments. The first argument identifies the type of filter (for example, an object), and the second argument specifies the value you are filtering on (for example, circles). The filter type is a DXF group code that specifies which filter to use. A few of the most common filter types are listed here.

For a complete list of DXF group codes, see “Group Codes in Numerical Order” in chapter 1, “DXF Format,” in the *DXF Reference*.

DXF codes for common filters

DXF code	Filter type
0	Object Type (String) Such as "Line," "Circle," "Arc," and so forth.
2	Object Name (String) The table (given) name of a named object.
8	Layer Name (String) Such as "Layer 0."
60	Object Visibility (Integer) Use 0 = visible, 1 = invisible.
62	Color Number (Integer) Numeric index values ranging from 0 to 256. Zero indicates the BYBLOCK. 256 indicates BYLAYER. A negative value indicates that the layer is turned off.
67	Model/paper space indicator (Integer) Use 0 or omitted = model space, 1 = paper space.

The filter arguments are declared as arrays. The filter type is declared as an integer and the filter value as a variant. Each filter type must be paired with a filter value. For example:

```
FilterType(0) = 0      ' Indicates filter refers to an object type
FilterData(0) = "Circle" ' Indicates the object type is "Circle"
```

Specifying a single selection criterion for a selection set

The following code prompts users to select objects to be included in a selection set, but only adds the selected object if it is a Circle:

```
Sub Ch4_FilterMtext()
    Dim sstext As AcadSelectionSet
    Dim FilterType(0) As Integer
    Dim FilterData(0) As Variant
    Set sstext = ThisDrawing.SelectionSets.Add("SS2")
    FilterType(0) = 0
    FilterData(0) = "Circle"
    sstext.SelectOnScreen FilterType, FilterData
End Sub
```

In the following example, only blue objects are allowed in the selection set:

```
Sub Ch4_FilterColor()  
    Dim sstext As AcadSelectionSet  
    Dim FilterType(0) As Integer  
    Dim FilterData(0) As Variant  
    Set sstext = ThisDrawing.SelectionSets.Add("SS3")  
    FilterType(0) = 62  
    FilterData(0) = acBlue  
    sstext.SelectOnScreen FilterType, FilterData  
End Sub
```

Specifying Multiple Criteria in a Selection Set Filter List

To specify multiple selection criteria, declare an array containing enough elements to represent each criterion, and assign each criterion to an element.

Selecting objects that meet three criteria

The following code specifies three criteria: the object must be blue, it must be a Circle, and it must reside on Layer 0. The code declares FilterType and FilterData as arrays of three elements, and assigns each criterion to an element:

```
Sub Ch4_FilterBlueCircleOnLayer0()  
    Dim sstext As AcadSelectionSet  
    Dim FilterType(2) As Integer  
    Dim FilterData(2) As Variant  
    Set sstext = ThisDrawing.SelectionSets.Add("SS4")  
  
    FilterType(0) = 0  
    FilterData(0) = "Circle"  
  
    FilterType(1) = 62  
    FilterData(1) = acBlue  
  
    FilterType(2) = 8  
    FilterData(2) = "0"  
  
    sstext.SelectOnScreen FilterType, FilterData  
End Sub
```

Adding Complexity to Your Filter List Conditions

When you specify multiple selection criteria, AutoCAD assumes the selected object must meet each criterion. But you can qualify your criteria in other ways. For numeric items, you can specify relational operations (for example, the radius of a circle must be *greater than or equal* to 5.0). And for all items, you can specify logical operations (for example, Text *or* Mtext).

Use a -4 DXF code to indicate a relational operator in your filter specification. Specify the operator as a string. The allowable relational operators are shown in the following table.

Relational operators for selection set filter lists

Operator	Description
"*"	Anything goes (always true)
"="	Equals
"! ="	Not equal to
" / ="	Not equal to
" < > "	Not equal to
" < "	Less than
" < = "	Less than or equal to
" > "	Greater than
" > = "	Greater than or equal to
" & "	Bitwise AND (integer groups only)
" & = "	Bitwise masked equals (integer groups only)

Logical operators in filter lists are also indicated by a -4 group code, and the operator is a string, but the operators must be paired. The opening operator is preceded by a less-than symbol (<), and the closing operator is followed by a greater-than symbol (>). The following table lists the logical operators allowed in selection set filtering.

Logical grouping operators for selection set filter lists		
Starting operator	Encloses	Ending operator
"<AND"	One or more operands	"AND>"
"<OR"	One or more operands	"OR>"
"<XOR"	Two operands	"XOR>"
"<NOT"	One operand	"NOT>"

Selecting a circle whose radius is greater than or equal to 5.0

The following code specifies that the selected object must be a circle whose radius is greater than or equal to 5.0:

```
Sub Ch4_FilterRelational ()
    Dim sstext As AcadSelectionSet
    Dim FilterType(2) As Integer
    Dim FilterData(2) As Variant
    Set sstext = ThisDrawing.SelectionSets.Add("SS5")

    FilterType(0) = 0
    FilterData(0) = "Circle"
    FilterType(1) = -4
    FilterData(1) = ">="
    FilterType(2) = 40
    FilterData(2) = 5#

    sstext.SelectOnScreen FilterType, FilterData
End Sub
```

Selecting either Text or Mtext

The following example specifies that either Text or Mtext objects can be selected:

```
Sub Ch4_FilterOrTest()  
    Dim sstext As AcadSelectionSet  
    Dim FilterType(3) As Integer  
    Dim FilterData(3) As Variant  
    Set sstext = ThisDrawing.SelectionSets.Add("SS6")  
    FilterType(0) = -4  
    FilterData(0) = "<or"  
    FilterType(1) = 0  
    FilterData(1) = "TEXT"  
    FilterType(2) = 0  
    FilterData(2) = "MTEXT"  
    FilterType(3) = -4  
    FilterData(3) = "or">  
  
    sstext.SelectOnScreen FilterType, FilterData  
End Sub
```

Selecting either Text or Mtext, if the object is also either blue or green

The following example specifies that either Text or Mtext can be selected, but the object must also be colored either blue or green:

```
Sub Ch4_OrAndOrTest()  
    Dim sstext As AcadSelectionSet  
    Dim FilterType(9) As Integer  
    Dim FilterData(9) As Variant  
    Set sstext = ThisDrawing.SelectionSets.Add("SS8")  
    FilterType(0) = -4  
    FilterData(0) = "<or"  
    FilterType(1) = 0  
    FilterData(1) = "TEXT"  
    FilterType(2) = 0  
    FilterData(2) = "MTEXT"  
    FilterType(3) = -4  
    FilterData(3) = "or">  
    FilterType(4) = -4  
    FilterData(4) = "<and"  
    FilterType(5) = -4  
    FilterData(5) = "<or"  
    FilterType(6) = 62  
    FilterData(6) = acBlue  
    FilterType(7) = 62  
    FilterData(7) = acGreen  
    FilterType(8) = -4  
    FilterData(8) = "or">  
    FilterType(9) = -4  
    FilterData(9) = "and">  
  
    sstext.SelectOnScreen FilterType, FilterData  
End Sub
```

Using Wild-Card Patterns in Selection Set Filter Criteria

Symbol names and strings in filter lists can include wild-card patterns.

The following table identifies the wild-card characters recognized by AutoCAD, and what each means in the context of a string:

Wild-card characters	
Character	Definition
# (pound)	Matches any single numeric digit
@ (at)	Matches any single alphabetic character
. (period)	Matches any single nonalphanumeric character
* (asterisk)	Matches any character sequence, including an empty one, and it can be used anywhere in the search pattern: at the beginning, middle, or end
? (question mark)	Matches any single character
~ (tilde)	If it is the first character in the pattern, it matches anything except the pattern
[. . .]	Matches any one of the characters enclosed
[~ . . .]	Matches any single character not enclosed
– (hyphen)	Used inside brackets to specify a range for a single character
, (comma)	Separates two patterns
` (reverse quote)	Escapes special characters (reads next character literally)

Use a reverse quote (‘) to indicate that a character is not a wild-card, but is to be taken literally. For example, to specify that only an anonymous block named “*U2” be included in the selection set, use the following filter arguments:

```
FilterType(0) = 2  
FilterData(0) = "‘ *U2"
```

Selecting Mtext where a specific word appears in the text

The following code defines the selection criteria as any Mtext in which “The” appears in the text string. This example also demonstrates use of the SelectByPolygon selection method:

```
Sub Ch4_FilterPolygonWildcard()  
    Dim sstext As AcadSelectionSet  
    Dim FilterType(1) As Integer  
    Dim FilterData(1) As Variant  
    Dim pointsArray(0 To 11) As Double  
    Dim mode As Integer  
    mode = acSelectionSetWindowPolygon  
    pointsArray(0) = -12#: pointsArray(1) = -7#: pointsArray(2) = 0  
    pointsArray(3) = -12#: pointsArray(4) = 10#: pointsArray(5) = 0  
    pointsArray(6) = 10#: pointsArray(7) = 10#: pointsArray(8) = 0  
    pointsArray(9) = 10#: pointsArray(10) = -7#: pointsArray(11) = 0  
    Set sstext = ThisDrawing.SelectionSets.Add("SS10")  
  
    FilterType(0) = 0  
    FilterData(0) = "MTEXT"  
    FilterType(1) = 1  
    FilterData(1) = "*The*"  
  
    sstext.SelectByPolygon mode, pointsArray, FilterType, FilterData  
End Sub
```

Filtering for Extended Data

External applications can attach data such as text strings, numeric values, 3D points, distances, and layer names to AutoCAD objects. This data is referred to as extended data, or xdata. You can filter entities containing extended data for a specified application.

See “Assigning and Retrieving Extended Data” on page 309 for more information about extended data.

Selecting circles that contain xdata

The following example filters for circles containing xdata added by the “MY_APP” application:

```

Sub Ch4_FilterXdata()
    Dim sstext As AcadSelectionSet
    Dim mode As Integer
    Dim pointsArray(0 To 11) As Double
    mode = acSelectionSetWindowPolygon
    pointsArray(0) = -12#: pointsArray(1) = -7#: pointsArray(2) = 0
    pointsArray(3) = -12#: pointsArray(4) = 10#: pointsArray(5) = 0
    pointsArray(6) = 10#: pointsArray(7) = 10#: pointsArray(8) = 0
    pointsArray(9) = 10#: pointsArray(10) = -7#: pointsArray(11) = 0
    Dim FilterType(1) As Integer
    Dim FilterData(1) As Variant
    Set sstext = ThisDrawing.SelectionSets.Add("SS9")

    FilterType(0) = 0
    FilterData(0) = "Circle"
    FilterType(1) = 1001
    FilterData(1) = "MY_APP"

    sstext.SelectByPolygon mode, pointsArray, FilterType, FilterData
End Sub

```

Displaying Information About a Selection Set

To refer to an existing selection set whose name you know, refer to it by name. The following example refers to a selection set named "SS10:"

```

Sub GetObjInSet()
    Dim selset As AcadSelectionSet
    Set selset = ThisDrawing.SelectionSets("SS10")

    MsgBox ("Selection set " & selset.Name & " contains " & _
        selset.Count & " items")
End Sub

```

Each selection set in a drawing is a member of the SelectionSets collection. You can use the For Each statement to iterate through a drawing's SelectionSets collection and collect information about each selection set.

Displaying the name of each selection set in a drawing

The following code displays the name of each selection set in a drawing, and lists the types of objects included in each selection set:

```
Sub ListSelectionSets()  
    Dim selSetCollection As AcadSelectionSets  
    Dim selSet As AcadSelectionSet  
    Dim ent As Object  
    Dim i, j As Integer  
  
    Set selSetCollection = ThisDrawing.SelectionSets  
  
    ' Find each selection set in the drawing  
    i = 0  
    For Each selSet In selSetCollection  
        MsgBox "Selection set " & CStr(i) & " is: " & selSet.Name  
  
        ' Now find each object in the selection set, and say what it is  
        j = 0  
        For Each ent In selSet  
            MsgBox "Item " & CStr(j + 1) & " in " & selSet.Name _  
                & "is: " & ent.EntityName  
            j = j + 1  
        Next  
        i = i + 1  
    Next  
  
End Sub
```

Removing Objects from a Selection Set

After you create a selection set, you can choose to remove individual objects, or all the objects from that set. For example, you can select an entire group of densely grouped objects and remove specific objects within the group, leaving only the objects you want to be in the set.

Use the following methods to remove items from the selection set:

- | | |
|--------------------|---|
| RemoveItems | The RemoveItems method removes one or more items from a selection set. The removed items still exist, but they no longer reside in the selection set. |
| Clear | The Clear method will empty the selection set. The selection set will still exist, but contains no items. The items that previously resided in the selection still exist, but they no longer reside in the selection set. |

Erase	The Erase method deletes all items in a selection set. The selection set still exists, but will contain no items. The items that previously resided in the selection set no longer exist.
Delete	The Delete method deletes a selection set and all items in the selection set. Neither the selection set, nor the items previously in the selection set will exist after a call to the Delete method.

Editing Objects

To modify an existing object, use the methods and properties associated with that object. If you modify a visible property of a graphic object, use the Update method to redraw the object onscreen.

This section describes how to edit 2D objects.

Working with Named Objects

In addition to the graphic objects used by AutoCAD, there are several types of nongraphic objects stored in drawing files. These objects have descriptive designations associated with them, for example, blocks, layers, groups, and dimension styles. In most cases, you name objects as you create them, and rename them later. Names are stored in symbol tables. When you specify a named object, you are referencing the name and associated data of the object in the symbol table.

Purging Named Objects

You can purge unused, unreferenced named objects from a drawing at any time during an editing session. Purging reduces drawing size. You cannot purge objects that are referenced by other objects. For example, a font file might be referenced by a text style. A layer is referenced by the objects on the layer.

To purge a drawing, use the PurgeAll method, as follows:

```
Thi sDrawi ng. PurgeAl l
```

Renaming Objects

As your drawings become more complex, you can rename objects to keep the names meaningful or to avoid conflicts with names in other drawings you have inserted in the main drawings.

You can rename any named object except those that AutoCAD names by default, for example, layer 0 or the CONTINUOUS linetype.

Names can be up to 255 characters long. In addition to letters and numbers, names can contain spaces (although AutoCAD removes spaces that appear directly before and after a name) and any special character not used by Microsoft Windows or AutoCAD for other purposes. Special characters that you cannot use include less-than and greater-than symbols (< >), forward slashes and backslashes (/ \), quotation marks ("), colons (:), semicolons (;), question marks (?), commas (,), asterisks (*), vertical bars (|), equal signs (=), and backquotes (`). You also cannot use special characters created with Unicode fonts.

To rename an object, use the [Name](#) property for that object.

Renaming a layer

This example creates a layer called “NewLayer” and then renames the layer to “MyLayer”.

```
Sub Ch4_RenamingLayer()  
    ' Create a layer  
    Dim LayerObj As AcadLayer  
    Set LayerObj = ThisDrawing.Layers.Add("NewLayer")  
  
    ' Change the name of the layer  
    LayerObj.Name = "MyLayer"  
  
End Sub
```

Copying Objects

You can copy single or multiple objects within the current drawing. Offsetting creates new objects at a specified distance from selected objects, or through a specified point. Mirroring creates a mirror image of objects in a specified mirror line. Arraying creates sets of copied objects in a rectangular or circular pattern.

For more information about copying objects, see “Copy, Offset, or Mirror Objects” in chapter 17, “Change Existing Objects,” in the *User's Guide*.

NOTE You cannot perform any of the copy methods while simultaneously iterating through a collection. An iteration will open the workspace for a read-only operation while these methods attempt to perform a read-write operation. Complete any iteration of a collection before you call these methods.

Copying an Object to the Same Location

To copy a single object, use the Copy method provided for that object. This method creates a new object that is a duplicate of the original object. The new object is located at the same position as the original, and is returned by the method.

Copying Multiple Objects

To copy multiple objects, use the CopyObjects method or create an array of objects to use with the Copy method. (To copy the objects in a selection set, iterate through the selection set and save the objects into an array.) Iterate through the array, copying each object individually, and collect the newly created objects in a second array.

Copying two circle objects

This example creates two Circle objects and uses the CopyObjects method to make a copy of the circles.

```
Sub Ch4_CopyCircleObjects()  
    Dim DOC1 As AcadDocument  
    Dim circleObj1 As AcadCircle  
    Dim circleObj2 As AcadCircle  
    Dim circleObj1Copy As AcadCircle  
    Dim circleObj2Copy As AcadCircle  
    Dim centerPoint(0 To 2) As Double  
    Dim radius1 As Double  
    Dim radius2 As Double  
    Dim radius1Copy As Double  
    Dim radius2Copy As Double  
    Dim objCollection(0 To 1) As Object  
    Dim retObjects As Variant  
  
    ' Define the Circle object  
    centerPoint(0) = 0: centerPoint(1) = 0: centerPoint(2) = 0  
    radius1 = 5#: radius2 = 7#  
    radius1Copy = 1#: radius2Copy = 2#  
  
    ' Create a new drawing  
    Set DOC1 = ThisDrawing.Application.Documents.Add
```

```

' Add two circles to the drawing
Set circleObj1 = DOC1.ModelSpace.AddCircle _
    (centerPoint, radius1)
Set circleObj2 = DOC1.ModelSpace.AddCircle _
    (centerPoint, radius2)

ZoomAll

' Put the objects to be copied into a form
' compatible with CopyObjects
Set objCollection(0) = circleObj1
Set objCollection(1) = circleObj2

' Copy object and get back a collection of
' the new objects (copies)
retObjects = DOC1.CopyObjects(objCollection)

' Get newly created object and apply
' new properties to the copies
Set circleObj1Copy = retObjects(0)
Set circleObj2Copy = retObjects(1)

circleObj1Copy.radius = radius1Copy
circleObj1Copy.Color = acRed
circleObj2Copy.radius = radius2Copy
circleObj2Copy.Color = acRed

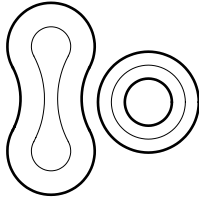
ZoomAll
End Sub

```

Offsetting Objects

Offsetting an object creates a new object at a specified offset distance from the original object. You can offset arcs, circles, ellipses, lines, lightweight polylines, polylines, splines, and xlines.

To offset an object, use the Offset method provided for that object. The only input to this method is the distance to offset the object. If this distance is negative, it is interpreted by AutoCAD as being an offset to make a “smaller” curve (that is, for an arc it would offset to a radius that is the given distance less than the starting curve’s radius). If “smaller” has no meaning, then AutoCAD would offset in the direction of smaller *X,Y,Z* WCS coordinates. If the offset distance is invalid then an error is returned.



offset polylines

For many objects, the result of this operation will be a single new curve (which may not be of the same type as the original curve). For example, offsetting an ellipse will result in a spline because the result does fit the equation of an ellipse. In some cases it may be necessary for the offset result to be several curves. Because of this, the method returns the new object, or array of objects, as a variant.

For more information about offsetting objects, see “Copy, Offset, or Mirror Objects” in chapter 17, “Change Existing Objects,” in the *User’s Guide*.

Offsetting a polyline

This example creates a lightweight polyline and then offsets the polyline. The newly offset polyline is colored red.

```
Sub Ch4_OffsetPolyline()
    ' Create the polyline
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 11) As Double
    points(0) = 1: points(1) = 1
    points(2) = 1: points(3) = 2
    points(4) = 2: points(5) = 2
    points(6) = 3: points(7) = 2
    points(8) = 4: points(9) = 4
    points(10) = 4: points(11) = 1
    Set plineObj = ThisDrawing.ModelSpace. _
        AddLightweightPolyline(points)
    plineObj.Closed = True
    ZoomAll

    ' Offset the polyline
    Dim offsetObj As Variant
    offsetObj = plineObj.Offset(0.25)
    offsetObj(0).Color = acRed

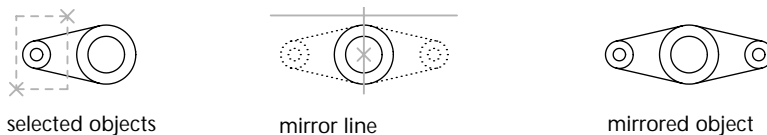
    ZoomAll
End Sub
```

Mirroring Objects

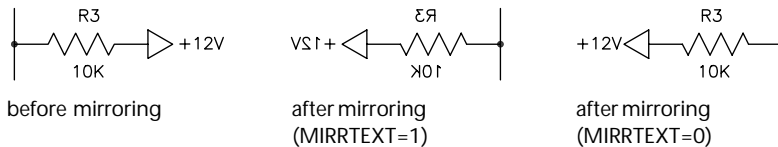
Mirroring creates a mirror image copy of an object around an axis or mirror line. You can mirror all drawing objects.

To mirror an object, use the Mirror method provided for that object. This method requires two coordinates as input. The two coordinates specified become the endpoints of the mirror line around which the base object is reflected. In 3D, this line orients a mirroring plane perpendicular to the *XY* plane of the UCS containing the mirror line.

Unlike the mirror command in AutoCAD, this method places the reflected image into the drawing and retains the original object. (To remove the original object, use the Erase method.)



To manage the reflection properties of Text objects, use the MIRRTEXT system variable. The default setting of MIRRTEXT is On (1), which causes Text objects to be mirrored just as any other object. When MIRRTEXT is off (0), text is not mirrored. Use the GetVariable and SetVariable methods to query and set the MIRRTEXT setting.



You can mirror a Viewport object in paper space, although doing so has no effect on its model space view or on model space objects.

For more information about mirroring objects, see “Copy, Offset, or Mirror Objects” in chapter 17, “Change Existing Objects,” in the *User's Guide*.

Mirroring a polyline about an axis

This example creates a lightweight polyline and mirrors that polyline about an axis. The newly created polyline is colored red.

```

Sub Ch4_MirrorPolyline()
    ' Create the polyline
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 11) As Double
    points(0) = 1: points(1) = 1
    points(2) = 1: points(3) = 2
    points(4) = 2: points(5) = 2
    points(6) = 3: points(7) = 2
    points(8) = 4: points(9) = 4
    points(10) = 4: points(11) = 1
    Set plineObj = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)
    plineObj.Closed = True
    ZoomAll

    ' Define the mirror axis
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    point1(0) = 0: point1(1) = 4.25: point1(2) = 0
    point2(0) = 4: point2(1) = 4.25: point2(2) = 0

    ' Mirror the polyline
    Dim mirrorObj As AcadLWPolyline
    Set mirrorObj = plineObj.Mirror(point1, point2)
    mirrorObj.Color = acRed

    ZoomAll
End Sub

```

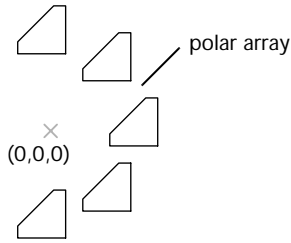
Arraying Objects

You can copy an object in polar or rectangular arrays. For polar arrays, you control the number of copies of the object and the angle to fill the array to. For rectangular arrays, you control the number of rows and columns and the distance between them.

For more information about arrays, see “Create an Array of Objects” in chapter 17, “Change Existing Objects,” in the *User’s Guide*.

Creating Polar Arrays

You can array all drawing objects. To create a polar array, use the `ArrayPolar` method provided for that object. This method requires you to provide the number of objects to create, the angle-to-fill, and the center point for the array. The number of objects must be a positive integer greater than 1. The angle-to-fill must be in radians. A positive value specifies counterclockwise rotation. A negative value specifies clockwise rotation. An error is returned for an angle that equals 0. The center point is a variant array containing three doubles. These doubles represent the 3D WCS coordinate specifying the center point for the polar array.



AutoCAD determines the distance from the array's center point to a reference point on the original object. The reference point used depends on the type of object. AutoCAD uses the center point of a circle or arc, the insertion point of a block or shape, the start point of text, and one endpoint of a line or trace.

This method does not support the Rotate While Copying option of the AutoCAD ARRAY command.

Creating a polar array

This example creates a circle, and then performs a polar array of the circle. This creates four circles filling 180 degrees around a base point of (4, 4, 0).

```
Sub Ch4_ArrayingACircle()
    ' Create the circle
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2#: center(1) = 2#: center(2) = 0#
    radius = 1
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)

    ZoomAll

    ' Define the polar array
    Dim noOfObjects As Integer
    Dim angleToFill As Double
    Dim basePnt(0 To 2) As Double
    noOfObjects = 4
    angleToFill = 3.14 ' 180 degrees
    basePnt(0) = 4#: basePnt(1) = 4#: basePnt(2) = 0#

    ' The following example will create 4 copies
    ' of an object by rotating and copying it about
    ' the point (3, 3, 0).
    Dim retObj As Variant
    retObj = circleObj.ArrayPolar _
        (noOfObjects, angleToFill, basePnt)

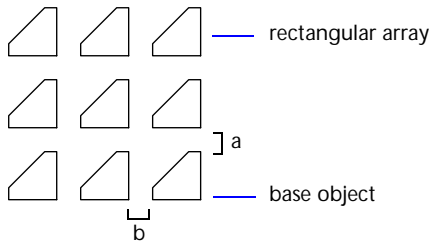
    ZoomAll
End Sub
```

Creating Rectangular Arrays

To create a 2D or 3D rectangular array, use the `ArrayRectangular` method provided for that object. This method requires you to provide the number of rows, number of columns, distance between rows, and distance between columns. When creating a 3D array, you must also specify the number of levels and distance between levels as well.

A rectangular array is constructed by replicating the object in the selection set the appropriate number of times. If you define one row, you must specify more than one column and vice versa.

The original object is assumed to be in the lower-left corner, and the array is generated up and to the right. If the distance between rows is a negative number, rows are added downward. If the distance between columns is a negative number, the columns are added to the left.



AutoCAD builds the rectangular array along a baseline defined by the current snap rotation angle. This angle is 0 by default, so the rows and columns of a rectangular array are orthogonal with respect to the *X* and *Y* drawing axes. You can change this angle and create a rotated array by setting the snap rotation angle to a nonzero value. To do this, use the `SnapRotationAngle` property.

Creating a rectangular array

This example creates a circle and then performs a rectangular array of the circle, creating five rows and five columns of circles.

```

Sub Ch4_ArrayRectangularExample()
    ' Create the circle
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2#: center(1) = 2#: center(2) = 0#
    radius = 0.5
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)

    ZoomAll

    ' Define the rectangular array
    Dim numberOfRows As Long
    Dim numberOfColumns As Long
    Dim numberOfLevels As Long
    Dim distanceBwtnRows As Double
    Dim distanceBwtnColumns As Double
    Dim distanceBwtnLevels As Double
    numberOfRows = 5
    numberOfColumns = 5
    numberOfLevels = 2
    distanceBwtnRows = 1
    distanceBwtnColumns = 1
    distanceBwtnLevels = 1

    ' Create the array of objects
    Dim retObj As Variant
    retObj = circleObj.ArrayRectangular _
        (numberOfRows, numberOfColumns, numberOfLevels, _
        distanceBwtnRows, distanceBwtnColumns, distanceBwtnLevels)

    ZoomAll
End Sub

```

Moving Objects

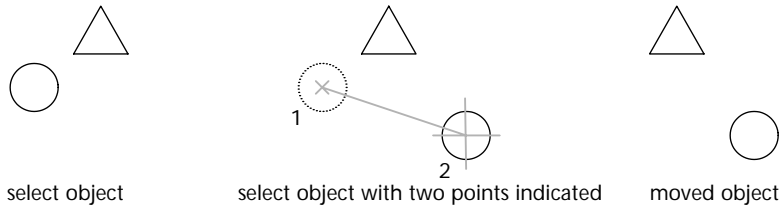
You can move objects along a vector without changing their orientation or size. You can also rotate objects around a base point.

For more information about moving objects, see “Move Objects” in chapter 17, “Change Existing Objects,” in the *User's Guide*.

Moving Objects Along a Vector

You can move all drawing objects and attribute reference objects along a specified vector.

To move an object, use the `Move` method provided for that object. This method requires two coordinates as input. These coordinates define a displacement vector indicating how far the given object is to be moved and in what direction.



Moving a circle along a vector

This example creates a circle and then moves that circle two units along the *X* axis.

```
Sub Ch4_MoveCircle()
' Create the circle
Dim circleObj As AcadCircle
Dim center(0 To 2) As Double
Dim radius As Double
center(0) = 2#: center(1) = 2#: center(2) = 0#
radius = 0.5
Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)

ZoomAll

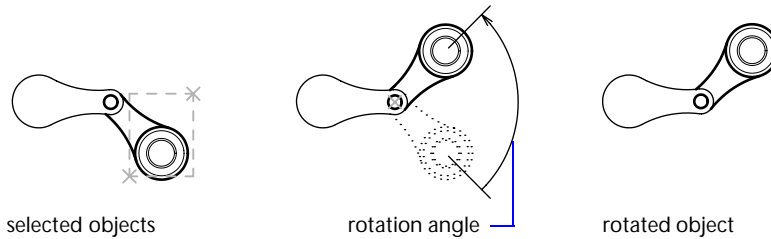
' Define the points that make up the move vector.
' The move vector will move the circle 2 units
' along the x axis.
Dim point1(0 To 2) As Double
Dim point2(0 To 2) As Double
point1(0) = 0: point1(1) = 0: point1(2) = 0
point2(0) = 2: point2(1) = 0: point2(2) = 0

' Move the circle
circleObj.Move point1, point2
circleObj.Update
End Sub
```

Rotating Objects

You can rotate all drawing objects and attribute reference objects.

To rotate an object, use the Rotate method provided for that object. This method requires as input a base point and a rotation angle. The base point is a variant array with three doubles. These doubles represent a 3D WCS coordinate specifying the point through which the axis of rotation is defined. The angle of rotation is specified in radians. This angle determines how far an object rotates around the base point relative to its current location.



For more information about rotating objects, see “Rotate Objects” in chapter 17, “Change Existing Objects,” in the *User’s Guide*.

Rotating a polyline about a base point

This example creates a closed lightweight polyline, and then rotates the polyline 45 degrees about the base point (4, 4.25, 0).

```
Sub Ch4_RotatePolyline()
' Create the polyline
Dim plineObj As AcadLWPolyline
Dim points(0 To 11) As Double
points(0) = 1: points(1) = 2
points(2) = 1: points(3) = 3
points(4) = 2: points(5) = 3
points(6) = 3: points(7) = 3
points(8) = 4: points(9) = 4
points(10) = 4: points(11) = 2
Set plineObj = ThisDrawing.ModelSpace.AddLightweightPolyline(points)
plineObj.Closed = True
ZoomAll

' Define the rotation of 45 degrees about a
' base point of (4, 4.25, 0)
Dim basePoint(0 To 2) As Double
Dim rotationAngle As Double
basePoint(0) = 4: basePoint(1) = 4.25: basePoint(2) = 0
rotationAngle = 0.7853981 ' 45 degrees

' Rotate the polyline
plineObj.Rotate basePoint, rotationAngle
plineObj.Update
End Sub
```

Deleting Objects

You can delete individual objects by using the `Delete` method.

NOTE The Collection objects in ActiveX Automation have a `Delete` method due to the manner in which these objects have been defined in the type library. However, the Collection objects, such as `ModelSpace` collection, `Layers` collection, and `Dictionaries` collection, should never be deleted. An error will result if you attempt to delete a collection.

Creating and deleting a polyline

This example creates a lightweight polyline, then deletes it.

```
Sub Ch4_DeletePolyline()  
    ' Create the polyline  
    Dim lwpolyobj As AcadLWPolyline  
    Dim vertices(0 To 5) As Double  
    vertices(0) = 2: vertices(1) = 4  
    vertices(2) = 4: vertices(3) = 2  
    vertices(4) = 6: vertices(5) = 4  
    Set lwpolyobj = ThisDrawing.ModelSpace.  
        AddLightweightPolyline(vertices)  
    ZoomAll  
  
    ' Erase the polyline  
    lwpolyobj.Delete  
    ThisDrawing.Regen acActiveViewport  
End Sub
```

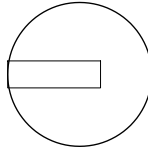
Scaling Objects

You scale an object by specifying a base point and a length, which is used as a scale factor based on the current drawing units. You can scale all the drawing objects, as well as attribute reference objects.

To scale an object, use the `ScaleEntity` method provided for that object. This method scales the object equally in the *X*, *Y*, and *Z* directions. It takes as input the base point for the scale and a scale factor. The base point is a variant array with three doubles. These doubles represent a 3D WCS coordinate specifying the point from which the scale begins. The scale factor is the factor by which to scale the object. The dimensions of the object are multiplied by the scale factor. A scale factor greater than 1 enlarges the object. A scale factor between 0 and 1 reduces the object.



scale factor = .5



scale factor = 2

For more information about scaling, see “Resize or Reshape Objects” in chapter 17, “Change Existing Objects,” in the *User’s Guide*.

Scaling a polyline

This example creates a closed lightweight polyline and then scales the polyline by 0.5.

```
Sub Ch4_ScalePolyline()
    ' Create the polyline
    Dim plineObj As AcadLWPolyline
    Dim points(0 To 11) As Double
    points(0) = 1: points(1) = 2
    points(2) = 1: points(3) = 3
    points(4) = 2: points(5) = 3
    points(6) = 3: points(7) = 3
    points(8) = 4: points(9) = 4
    points(10) = 4: points(11) = 2
    Set plineObj = ThisDrawing.ModelSpace.AddLightweightPolyline(points)
    plineObj.Closed = True
    ZoomAll

    ' Define the scale
    Dim basePoint(0 To 2) As Double
    Dim scaleFactor As Double
    basePoint(0) = 4: basePoint(1) = 4.25: basePoint(2) = 0
    scaleFactor = 0.5

    ' Scale the polyline
    plineObj.ScaleEntity basePoint, scaleFactor
    plineObj.Update
End Sub
```

Transforming Objects

You move, scale, or rotate an object given a 4×4 transformation matrix using the TransformBy method.

The following table demonstrates the transformation matrix configuration, where R = Rotation and T = Translation:

Transformation matrix configuration			
R00	R01	R02	T0
R10	R11	R12	T1
R20	R21	R22	T2
0	0	0	1

To transform an object, first initialize the transformation matrix. The following example shows a transformation matrix, assigned to the variable tMatrix, which will rotate an entity by 90 degrees about the point (0, 0, 0):

```
tMatrix(0, 0) = 0.0
tMatrix(0, 1) = -1.0
tMatrix(0, 2) = 0.0
tMatrix(0, 3) = 0.0
tMatrix(1, 0) = 1.0
tMatrix(1, 1) = 0.0
tMatrix(1, 2) = 0.0
tMatrix(1, 3) = 0.0
tMatrix(2, 0) = 0.0
tMatrix(2, 1) = 0.0
tMatrix(2, 2) = 1.0
tMatrix(2, 3) = 0.0
tMatrix(3, 0) = 0.0
tMatrix(3, 1) = 0.0
tMatrix(3, 2) = 0.0
tMatrix(3, 3) = 1.0
```

After the transformation matrix is complete, apply the matrix to the object using the TransformBy method. The following line of code demonstrates applying a matrix (tMatrix) to an object (anObj):

```
anObj.TransformBy tMatrix
```

Rotating a line using a transformation matrix

This example creates a line and rotates it 90 degrees using a transformation matrix.

```
Sub Ch4_TransformBy()
    ' Create a line
    Dim lineObj As AcadLine
    Dim startPt(0 To 2) As Double
    Dim endPt(0 To 2) As Double
    startPt(0) = 2
    startPt(1) = 1
    startPt(2) = 0
    endPt(0) = 5
    endPt(1) = 1
    endPt(2) = 0
    Set lineObj = ThisDrawing.ModelSpace.AddLine(startPt, endPt)

    ZoomAll

    ' Initialize the transMat variable with a
    ' transformation matrix that will rotate
    ' an object by 90 degrees about the point(0,0,0)
    Dim transMat(0 To 3, 0 To 3) As Double
    transMat(0, 0) = 0#: transMat(0, 1) = -1#
    transMat(0, 2) = 0#: transMat(0, 3) = 0#
    transMat(1, 0) = 1#: transMat(1, 1) = 0#
    transMat(1, 2) = 0#: transMat(1, 3) = 0#
    transMat(2, 0) = 0#: transMat(2, 1) = 0#
    transMat(2, 2) = 1#: transMat(2, 3) = 0#
    transMat(3, 0) = 0#: transMat(3, 1) = 0#
    transMat(3, 2) = 0#: transMat(3, 3) = 1#

    ' Transform the line using the defined transformation matrix
    lineObj.TransformBy transMat
    lineObj.Update
End Sub
```

The following are more examples of transformation matrices:

Rotation Matrix: 90 degrees about point (0, 0, 0)			
0.0	-1.0	0.0	0.0
1.0	0.0	0.0	0.0
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0

Rotation Matrix: 45 degrees about point (5, 5, 0)			
0.707107	-0.707107	0.0	5.0
0.707107	0.707107	0.0	-2.071068
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0

Translation Matrix: move an entity by (10, 10, 0)			
1.0	0.0	0.0	10.0
0.0	1.0	0.0	10.0
0.0	0.0	1.0	0.0
0.0	0.0	0.0	1.0

Scaling Matrix: scale by 10,10 at point (0, 0, 0)			
10.0	0.0	0.0	0.0
0.0	10.0	0.0	0.0
0.0	0.0	10.0	0.0
0.0	0.0	0.0	1.0

Scaling Matrix: scale by 10,10 at point (2, 2, 0)			
10.0	0.0	0.0	-18.0
0.0	10.0	0.0	-18.0
0.0	0.0	10.0	0.0
0.0	0.0	0.0	1.0

Extending and Trimming Objects

You can change the angle of arcs and you can change the length of open lines, arcs, open polylines, elliptical arcs, and open splines. The results are similar to both extending and trimming objects.

You can extend or trim an object by editing its properties. For example, to lengthen a line, simply change the coordinates of the StartPoint or EndPoint properties. To change the angle of an arc, change the StartAngle or EndAngle properties of the arc. Once you have altered an object's property or properties, use the Update method to see your changes in the drawing.

For more information about extending and trimming objects, see “Resize or Reshape Objects” in chapter 17, “Change Existing Objects,” in the *User's Guide*.

Lengthening a line

This example creates a line then changes the endpoint of that line resulting in a longer line.

```
Sub Ch4_LengthenLine()  
    ' Define and create the line  
    Dim lineObj As AcadLine  
    Dim startPoint(0 To 2) As Double  
    Dim endPoint(0 To 2) As Double  
    startPoint(0) = 0  
    startPoint(1) = 0  
    startPoint(2) = 0  
    endPoint(0) = 1  
    endPoint(1) = 1  
    endPoint(2) = 1  
    Set lineObj = ThisDrawing.ModelSpace.AddLine(startPoint, endPoint)  
    lineObj.Update  
  
    ' Lengthen the line by changing the  
    ' endpoint to 4, 4, 4  
    endPoint(0) = 4  
    endPoint(1) = 4  
    endPoint(2) = 4  
    lineObj.EndPoint = endPoint  
    lineObj.Update  
End Sub
```

Exploding Objects

Exploding objects converts the objects from single objects to their constituent parts but has no visible effect. For example, exploding forms simple lines and arcs from 3D polygons, polylines, polygon meshes, and regions. It

replaces a block reference with copies of the simple objects that compose the block.

For more information about exploding objects, see “Disassociate Compound Objects (Explode)” in chapter 17, “Change Existing Objects,” in the *User’s Guide*.

Exploding a polyline

This example creates a lightweight polyline object. It then explodes the polyline into separate objects. The example then loops through the resulting objects and displays a message box containing the name of each object and its index in the list of exploded objects.

```
Sub Ch4_ExplodePolyline()  
    Dim plineObj As AcadLWPolyline  
    Dim points(0 To 11) As Double  
  
    ' Define the 2D polyline points  
    points(0) = 1: points(1) = 1  
    points(2) = 1: points(3) = 2  
    points(4) = 2: points(5) = 2  
    points(6) = 3: points(7) = 2  
    points(8) = 4: points(9) = 4  
    points(10) = 4: points(11) = 1  
  
    ' Create a lightweight Polyline object  
    Set plineObj = ThisDrawing.ModelSpace.AddLightWeightPolyline(points)  
  
    ' Set the bulge on one segment to vary the  
    ' type of objects in the polyline  
    plineObj.SetBulge 3, -0.5  
    plineObj.Update  
  
    ' Explode the polyline  
    Dim explodedObjects As Variant  
    explodedObjects = plineObj.Explode  
  
    ' Loop through the exploded objects  
    ' and displays a message box with  
    ' the type of each object  
    Dim I As Integer  
    For I = 0 To UBound(explodedObjects)  
        explodedObjects(I).Color = acRed  
        explodedObjects(I).Update  
        MsgBox "Exploded Object " & I & ": " & _  
            explodedObjects(I).ObjectName  
        explodedObjects(I).Color = acByLayer  
        explodedObjects(I).Update  
    Next  
End Sub
```

Editing Polylines

2D and 3D polylines, rectangles, polygons, and 3D polygon meshes are all polyline variants and are edited in the same way.

AutoCAD recognizes both fit polylines and spline-fit polylines. A spline-fit polyline uses a curve fit, similar to a B-spline. There are two kinds of spline-fit polylines: quadratic and cubic. Both polylines are controlled by the `SPLINETYPE` system variable. A fit polyline uses standard curves for curve fit and utilizes any tangent directions set on any given vertex.

To edit a polyline, use the properties and methods of the `LightweightPolyline` or `Polyline` objects. Use the following properties and methods to open or close a polyline, change the coordinates of a polyline vertex, or add a vertex:

<code>Closed</code> property	Opens or closes the polyline.
<code>Coordinates</code> property	Specifies the coordinates for each vertex in the polyline.
<code>AddVertex</code> method	Adds a vertex to a lightweight polyline.

Use the following methods to update the bulge or width of a polyline:

<code>SetBulge</code>	Sets the bulge of a polyline, given the segment index.
<code>SetWidth</code>	Sets the start and end width of a polyline, given the segment index.

For more information about editing polylines, see “Modify or Join Polylines” in chapter 17, “Change Existing Objects,” in the *User’s Guide*.

Editing a polyline

This example creates a lightweight polyline. It then adds a bulge to the third segment of the polyline, appends a vertex to the polyline, changes the width of the last segment, and finally closes the polyline.

```

Sub Ch4_EditPolyLine()
    Dim plineObj As AcadLWPolyLine
    Dim points(0 To 9) As Double

    ' Define the 2D polyline points
    points(0) = 1: points(1) = 1
    points(2) = 1: points(3) = 2
    points(4) = 2: points(5) = 2
    points(6) = 3: points(7) = 2
    points(8) = 4: points(9) = 4

    ' Create a light weight Polyline object
    Set plineObj = ThisDrawing.ModelSpace.AddLightWeightPolyLine(points)

    ' Add a bulge to segment 3
    plineObj.SetBulge 3, -0.5

    ' Define the new vertex
    Dim newVertex(0 To 1) As Double
    newVertex(0) = 4: newVertex(1) = 1

    ' Add the vertex to the polyline
    plineObj.AddVertex 5, newVertex

    ' Set the width of the new segment
    plineObj.SetWidth 4, 0.1, 0.5

    ' Close the polyline
    plineObj.Closed = True
    plineObj.Update
End Sub

```

Editing Splines

Use the following editable properties to change splines:

Closed	Opens or closes the spline.
ControlPoints	Specifies the control points of a spline.
EndTangent	Specifies the end tangent of the spline as a directional vector.
FitPoints	Specifies all the fit points of a spline.
FitTolerance	Refits the spline to the existing points with new tolerance values.
Knots	Specifies the knots vector for the spline.
StartTangent	Specifies the start tangent for the spline.

In addition, you can use the following methods to edit splines:

AddFitPoint	Adds a single fit point to the spline at a given index.
DeleteFitPoint	Deletes the fit point of a spline at a given index.
ElevateOrder	Elevates the order of the spline to the given order.
GetFitPoint	Gets the fit point of the spline at a given index. (Gets one fit point only. To query all the fit points of the spline, use the FitPoints property.)
Reverse	Reverses the direction of a spline.
SetControlPoint	Sets the control point of the spline at a given index.
SetFitPoint	Sets the fit point of the spline at a given index. (Sets one fit point only. To change all the fit points of the spline, use the FitPoints property.)
SetWeight	Sets the weight of the control point at a given index.

Use the following read-only properties to query splines:

Degree	Gets the degree of the spline's polynomial representation.
Area	Gets the enclosed area of a spline.
IsPeriodic	Specifies if the given spline is periodic.
IsPlanar	Specifies if the given spline is planar.
IsRational	Specifies if the given spline is rational.
NumberOfControlPoints	Gets the number of control points of the spline.
NumberOfFitPoints	Gets the number of fit points of the spline.

For more information about editing splines, see “Modify Splines” in chapter 17, “Change Existing Objects,” in the *User's Guide*.

Changing a control point on a spline

This example creates a spline and then changes the first control point for the spline.

```
Sub Ch4_ChangeSplineControlPoint()  
    ' Create the spline  
    Dim splineObj As AcadSpline  
    Dim startTan(0 To 2) As Double  
    Dim endTan(0 To 2) As Double  
    Dim fitPoints(0 To 8) As Double  
  
    startTan(0) = 0.5: startTan(1) = 0.5: startTan(2) = 0  
    endTan(0) = 0.5: endTan(1) = 0.5: endTan(2) = 0  
    fitPoints(0) = 1: fitPoints(1) = 1: fitPoints(2) = 0  
    fitPoints(3) = 5: fitPoints(4) = 5: fitPoints(5) = 0  
    fitPoints(6) = 10: fitPoints(7) = 0: fitPoints(8) = 0  
    Set splineObj = ThisDrawing.ModelSpace.  
        AddSpline(fitPoints, startTan, endTan)  
    splineObj.Update  
  
    ' Change the coordinate of the first fit point  
    Dim controlPoint(0 To 2) As Double  
    controlPoint(0) = 0  
    controlPoint(1) = 3  
    controlPoint(2) = 0  
    splineObj.SetControlPoint 0, controlPoint  
    splineObj.Update  
End Sub
```

Editing Hatches

You can edit both hatch boundaries and hatch patterns. If you edit the boundary of an associative hatch, the pattern is updated as long as the editing results in a valid boundary. Associative hatches are updated even if they're on layers that are turned off. You can modify hatch patterns or choose a new pattern for an existing hatch, but associativity can only be set when a hatch is created. You can check to see if a Hatch object is associative by using the [AssociativeHatch](#) property. (See the AddHatch method for more information on creating a hatch.)

You must re-evaluate a hatch using the Evaluate method to see any edits to the hatch.

For more information about editing hatches, see “Modify Hatches and Solid-Filled Areas” in chapter 17, “Change Existing Objects,” in the *User's Guide*.

Editing Hatch Boundaries

You can append or insert loops into the hatch boundaries. Associative hatches are updated to match any changes made to their boundaries. Non-associative hatches are not updated.

To edit a hatch boundary, use one of the following methods:

AppendInnerLoop

Appends an inner loop to the hatch.

AppendOuterLoop

Appends an outer loop to the hatch.

InsertLoopAt

Inserts a loop at a given index of a hatch.

Appending an inner loop to a hatch

This example creates an associative hatch. It then creates a circle and appends the circle as an inner loop to the hatch.

```
Sub Ch4_AppendInnerLoopToHatch()  
    Dim hatchObj As AcadHatch  
    Dim patternName As String  
    Dim PatternType As Long  
    Dim bAssociativity As Boolean  
  
    ' Define and create the hatch  
    patternName = "ANSI 31"  
    PatternType = 0  
    bAssociativity = True  
    Set hatchObj = ThisDrawing.ModelSpace.AddHatch(PatternType, patternName, bAssociativity)
```

```

' Create the outer loop for the hatch.
Dim outerLoop(0 To 1) As AcadEntity
Dim center(0 To 2) As Double
Dim radius As Double
Dim startAngle As Double
Dim endAngle As Double
center(0) = 5: center(1) = 3: center(2) = 0
radius = 3
startAngle = 0
endAngle = 3.141592
Set outerLoop(0) = ThisDrawing.ModelSpace. _
    AddArc(center, radius, startAngle, endAngle)
Set outerLoop(1) = ThisDrawing.ModelSpace. _
    AddLine(outerLoop(0).startPoint, outerLoop(0).endPoint)

' Append the outer loop to the hatch object
hatchObj.AppendOuterLoop (outerLoop)

' Create a circle as the inner loop for the hatch.
Dim innerLoop(0) As AcadEntity
center(0) = 5: center(1) = 4.5: center(2) = 0
radius = 1
Set innerLoop(0) = ThisDrawing.ModelSpace. _
    AddCircle(center, radius)

' Append the circle as an inner loop to the hatch
hatchObj.AppendInnerLoop (innerLoop)

' Evaluate and display the hatch
hatchObj.Evaluate
ThisDrawing.Regen True
End Sub

```

Editing Hatch Patterns

You can change the angle or spacing of an existing hatch pattern or replace it with a solid-fill or one of the predefined patterns that AutoCAD offers. The Pattern option in the Boundary Hatch dialog box displays a list of these patterns. To reduce file size, the hatch is defined in the drawing as a single graphic object.

Use the following properties and methods to edit the hatch patterns:

PatternAngle	Specifies the angle of the hatch pattern.
PatternDouble	Specifies if the user-defined hatch is double-hatched.
PatternName	Specifies the hatch pattern name (does not change the pattern type).
PatternScale	Specifies the hatch pattern scale.
PatternSpace	Specifies the user-defined hatch pattern spacing.
SetPattern	Sets the pattern name and pattern type for the hatch.

Changing the pattern spacing of a hatch

This example creates a hatch. It then adds two to the current pattern spacing for the hatch.

```
Sub Ch4_ChangeHatchPatternSpace()  
    Dim hatchObj As AcadHatch  
    Dim patternName As String  
    Dim PatternType As Long  
    Dim bAssociativity As Boolean  
  
    ' Define the hatch  
    patternName = "ANSI31"  
    PatternType = 0  
    bAssociativity = True  
  
    ' Create the associative Hatch object  
    Set hatchObj = ThisDrawing.ModelSpace.AddHatch(PatternType, patternName, bAssociativity)  
  
    ' Create the outer loop for the hatch.  
    Dim outerLoop(0 To 0) As AcadEntity  
    Dim center(0 To 2) As Double  
    Dim radius As Double  
    center(0) = 5  
    center(1) = 3  
    center(2) = 0  
    radius = 3  
    Set outerLoop(0) = ThisDrawing.ModelSpace.AddCircle(center, radius)  
    hatchObj.AppendOuterLoop(outerLoop)  
    hatchObj.Evaluate  
  
    ' Change the spacing of the hatch pattern by  
    ' adding 2 to the current spacing  
    hatchObj.patternSpace = hatchObj.patternSpace + 2  
    hatchObj.Evaluate  
    ThisDrawing.Regen True  
End Sub
```


Using Layers, Colors, and Linetypes

Layers are like transparent overlays on which you organize and group different kinds of drawing information. The objects you create have properties including layers, colors, and linetypes. Color helps you distinguish similar elements in your drawings, and linetypes help you differentiate easily between different drafting elements, such as centerlines or hidden lines. Organizing layers and objects on layers makes it easier to manage the information in your drawings.

For more information about this topic, see chapter 14, “Control the Properties of Objects,” in the *User's Guide*.

Working with Layers

You are always drawing on a layer. It may be the default layer or a layer you create and name yourself. Each layer has an associated color and linetype. For example, you can create a layer on which you draw only centerlines and assign the color blue and the linetype CENTER to that layer. Then, whenever you want to draw centerlines you can switch to that layer and start drawing.

All layers and linetypes are kept within their parent Collection objects. Layers are kept within the Layers collection, and linetypes are kept within the Linetypes collection.

For more information about working with layers, see “Work with Layers” in chapter 14, “Control the Properties of Objects,” in the *User's Guide*.

Sorting Layers and Linetypes

You can iterate through the Layers and Linetypes collections to find all the layers and linetypes in a drawing.

Iterating through the Layers collection

The following code iterates through the Layers collection to gather the names of all the layers in the drawing. The names are then displayed in a message box.

```

Sub Ch4_IteratingLayers()
    Dim LayerNames As String
    Dim entry As AcadLayer
    LayerNames = ""
    For Each entry In ThisDrawing.Layers
        LayerNames = LayerNames + entry.Name + vbCrLf
    Next
    MsgBox "The layers in this drawing are: " + _
        vbCrLf + LayerNames
End Sub

```

Creating and Naming Layers

You can create new layers and assign color and linetype properties to those layers. Each individual layer is part of the Layers collection. Use the Add method to create a new layer and add it to the Layers collection.

You can assign a name to a layer when it is created. To change the name of a layer after it has been created, use the [Name](#) property. Layer names can include up to thirty-one characters and contain letters, digits, and the special characters dollar sign (\$), hyphen (-), and underscore (_) but cannot include blank spaces.

For more information about creating layers, see “Create and Name Layers” in chapter 14, “Control the Properties of Objects,” in the *User’s Guide*.

Create a new layer, assign it the color red, and add an object to the layer

The following code creates a circle and a new layer. The new layer is assigned the color red. The circle is assigned to the layer, and the color of the circle changes accordingly.

```

Sub Ch4_NewLayer()
' Create a circle
Dim circleObj As AcadCircle
Dim center(0 To 2) As Double
Dim radius As Double
center(0) = 2: center(1) = 2: center(2) = 0
radius = 1
Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)

' Assign the circle the color "ByLayer" so
' that the circle will automatically pick
' up the color of the layer on which it resides
circleObj.Color = acByLayer

' Create a new layer called "ABC"
Dim layerObj As AcadLayer
Set layerObj = ThisDrawing.Layers.Add("ABC")

' Assign the "ABC" layer the color red
layerObj.Color = acRed

' Assign the circle to the "ABC" layer
circleObj.Layer = "ABC"
circleObj.Update
End Sub

```

Making a Layer Active

You are always drawing on the active layer. When you make a layer active, you can create new objects on that layer. If you make a different layer active, any new objects you create are created on that new active layer and use its color and linetype. You cannot make a layer active if it is frozen.

To make a layer active, use the [ActiveLayer](#) property. This property is set on the current drawing. For example

```
Dim newLayer As AcadLayer
Set newLayer = ThisDrawing.Layers.Add("LAYER1")
ThisDrawing.ActiveLayer = newLayer
```

Turning Layers On and Off

Turned-off layers are regenerated with the drawing but are not displayed or plotted. By turning layers off, you avoid regenerating the drawing every time you thaw a layer. When you turn a layer on that has been turned off, AutoCAD redraws the objects on that layer.

To turn layers on and off, use the [LayerOn](#) property. If you input a value of TRUE to this property, the layer is turned on. If you input a value of FALSE, the layer is turned off.

For more information about turning layers on and off, see “Control the Visibility of Objects on a Layer” in chapter 14, “Control the Properties of Objects,” in the *User’s Guide*.

Turning off a layer

This example creates a new layer, adds a circle to the layer, then turns off the layer so that the circle is no longer visible.

```
Sub Ch4_LayerInvisible()
    ' Create a circle
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2: center(1) = 2: center(2) = 0
    radius = 1
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)
    circleObj.Color = acByLayer
```

```

' Create a new layer called "ABC"
Dim LayerObj As AcadLayer
Set LayerObj = ThisDrawing.Layers.Add("ABC")
LayerObj.Color = acRed

' Assign the circle to the "ABC" layer
circleObj.Layer = "ABC"
circleObj.Update

' Turn off layer "ABC"
LayerObj.LayerOn = False
ThisDrawing.Regen acActiveViewport
End Sub

```

Freezing and Thawing Layers

You can freeze layers to speed up display changes, improve object selection performance, and reduce regeneration time for complex drawings. AutoCAD does not display, plot, or regenerate objects on frozen layers. Freeze layers that you want to be invisible for long periods. When you “thaw” a frozen layer, AutoCAD regenerates and displays the objects on that layer.

To freeze or thaw a layer, use the [Freeze](#) property. If you input a value of TRUE to this property, the layer is frozen. If you input a value of FALSE, the layer is thawed.

For more information about freezing and thawing layers, see “Control the Visibility of Objects on a Layer” in chapter 14, “Control the Properties of Objects,” in the *User’s Guide*.

Freezing a layer

This example creates a new layer called “ABC” and then freezes the layer.

```

Sub Ch4_LayerFreeze()
' Create a new layer called "ABC"
Dim LayerObj As AcadLayer
Set LayerObj = ThisDrawing.Layers.Add("ABC")

' Freeze layer "ABC"
LayerObj.Freeze = True
End Sub

```

Locking and Unlocking Layers

You cannot edit the objects on a locked layer; however, they are still visible if the layer is on and thawed. You can make a locked layer current and you can add objects to it. You can freeze and turn off locked layers and change their associated colors and linetypes.

To lock or unlock a layer, use the [Lock](#) property. If you input a value of `TRUE` to this property, the layer is locked. If you input a value of `FALSE`, the layer is unlocked.

For more information about locking and unlocking layers, see “Control Whether Objects on a Layer Can Be Modified” in chapter 14, “Control the Properties of Objects,” in the *User’s Guide*.

Locking a layer

This example creates a new layer called “ABC” and then locks the layer.

```
Sub Ch4_LayerLock()  
    ' Create a new layer called "ABC"  
    Dim LayerObj As AcadLayer  
    Set LayerObj = ThisDrawing.Layers.Add("ABC")  
  
    ' Lock layer "ABC"  
    LayerObj.Lock = True  
End Sub
```

Assigning Color to a Layer

You can assign color to a layer. When specifying a color, you can enter the name of the color or its AutoCAD Color Index (ACI) number. Standard color names are available only for colors 1 to 7.

By default, AutoCAD assigns color number 7 (white or black, depending on your drawing background) to newly created layers. You can assign an object a color that is different from the layer color.

To assign color to a layer, use the [Color](#) property.

Colors can be set and read as numeric index values ranging from 0 to 256. Constants have been provided for the standard seven colors, and `BYBLOCK` and `BYLAYER` designations.

If you use `acByBlock`, AutoCAD draws new objects in the default color (white or black, depending on your configuration) until they are grouped into the block. When the block is inserted in the drawing, the objects in the block inherit the current setting of the `Color` property.

If you use `acByLayer`, new objects assume the color of the layer upon which they are drawn.

Assigning a Linetype to a Layer

When you're defining layers, linetypes provide another way to convey visual information. A linetype is a repeating pattern of dashes, dots, and blank spaces you can use to distinguish the purpose of one line from another.

The linetype name and definition describe the particular dash-dot sequence, the relative lengths of dashes and blank spaces, and the characteristics of any included text or shapes.

To assign a linetype to a layer, use the `Linetype` property. This property takes the name of the linetype as input.

See also “Working with Linetypes” on page 140.

Deleting Layers

To delete a layer, use the `Delete` method.

You can delete a layer at any time during a drawing session. You cannot delete the current layer, layer 0, an xref-dependent layer, or a layer that contains objects.

NOTE Layers referenced by block definitions, along with the special layer named `DEFPOINTS`, cannot be deleted even if they do not contain visible objects.

Working with Colors

You can assign colors to layers and to individual objects in a drawing. Each color is identified by a name or an AutoCAD Color Index (ACI) number, an integer from 1 through 255. Any number of objects and layers can have the same color number. You can assign each color number to a different pen on a pen plotter or use the color numbers to identify certain objects in the drawing, even though you can't see the colors on your screen.

When specifying a color, you can enter the name of the color or its ACI number. The ACI provides 255 color numbers. Standard color names are available only for colors 1 to 7.

Colors 1 to 7	
Color number	Color name
1	Red
2	Yellow
3	Green
4	Cyan
5	Blue
6	Magenta
7	Black/White

Colors 8 to 255 must be assigned by a number or by selecting the color in a dialog box. The default color (7) is either white or black, depending on your background color.

To set the current color of an object or a layer, use the [Color](#) property.

For more information about working with colors, see “Work with Colors” in chapter 14, “Control the Properties of Objects,” in the *User’s Guide*.

Working with Linetypes

A linetype is a repeating pattern of dashes, dots, and blank spaces. A complex linetype is a repeating pattern of symbols. To use a linetype you must first load it into your drawing. A linetype definition must exist in a LIN library file before a linetype can be loaded into a drawing. To load a linetype into your drawing, use the Load method.

For more information about working with linetypes, see “Overview of Linetypes” in chapter 14, “Control the Properties of Objects,” in the *User’s Guide*.

NOTE The linetypes used internally by AutoCAD should not be confused with the hardware linetypes provided by some plotters. The two types of dashed lines produce similar results. Do not use both types at the same time, however, because the results can be unpredictable.

Loading a linetype into AutoCAD

This example attempts to load the linetype “CENTER” from the *acad.lin* file. If the linetype already exists, or the file does not exist, then a message is displayed.

```
Sub Ch4_LoadLinetype()  
    On Error GoTo ERRORHANDLER  
  
    Dim LinetypeName As String  
    LinetypeName = "CENTER"  
  
    ' Load "CENTER" line type from acad.lin file  
    ThisDrawing.Linetypes.Load LinetypeName, "acad.lin"  
    Exit Sub  
  
ERRORHANDLER:  
    MsgBox Err.Description  
  
End Sub
```

Making a Linetype Active

To use a linetype to draw on the current layer, you must make it active. All newly created objects are drawn using the active linetype.

NOTE Xref-dependent linetypes cannot be made active.

To make a linetype active, use the [ActiveLinetype](#) property. This property is set on the current drawing. For example

```
ThisDrawing.ActiveLinetype = ThisDrawing.  
    Linetypes.Item("CONTINUOUS")
```

For more information about activating a linetype, see “Set the Current Linetype” in chapter 14, “Control the Properties of Objects,” in the *User’s Guide*.

Renaming Linetypes

To rename a linetype, use the [Name](#) property. When you rename a linetype, you are renaming only the linetype definition in your drawing. The name in the LIN library file is not being updated to reflect the new name.

Deleting Linetypes

To delete a linetype, use the `Delete` method. You can delete a linetype at any time during a drawing session; however, linetypes that cannot be deleted include `BYLAYER`, `BYBLOCK`, `CONTINUOUS`, the current linetype, and xref-dependent linetypes. Also, linetypes referenced by block definitions cannot be deleted, even if they are not used by any objects.

For more information about deleting linetypes, see “Load Linetypes” in chapter 14, “Control the Properties of Objects,” in the *User’s Guide*.

Changing Linetype Descriptions

Linetypes can have a description associated with them. The description provides an ASCII representation of the linetype. You can assign or change a linetype description by using the [Description](#) property.

A linetype description can have up to forty-seven characters. The description can be a comment or a series of underscores, dots, dashes, and spaces to show a simple representation of the linetype pattern. For example

```
ThisDrawing.ActiveLinetype.Description = "Exterior Wall I"
```

Specifying Linetype Scale

You can specify the linetype scale for objects you create. The smaller the scale, the more repetitions of the pattern are generated per drawing unit. By default, AutoCAD uses a global linetype scale of 1.0, which is equal to one drawing unit. You can change the linetype scale for all drawing objects, attribute references, and groups.

To change the linetype scale, use the [LinetypeScale](#) property.

The `CELTSKALE` system variable sets the linetype scale for newly created objects. `LTSCALE` globally changes the linetype scale of existing objects as well as new objects. To change the values of system variables using AutoCAD ActiveX Automation, use the `SetVariable` method.

For more information about linetype scales, see “Control Linetype Scale” in chapter 14, “Control the Properties of Objects,” in the *User’s Guide*.

Changing the linetype scale for a circle

```
Sub Ch4_ChangeLinetypeScale()  
  
    ' Save the current linetype  
    Set currLinetype = ThisDrawing.ActiveLinetype  
  
    ' Change the active linetype to Border, so the scale change will  
    ' be visible.  
    ' First see if the Border linetype is already loaded  
    On Error Resume Next  
    ThisDrawing.ActiveLinetype = ThisDrawing.Linetypes.Item("BORDER")  
    If Err.Number = -2145386476 Then  
        ' Error indicates linetype is not currently loaded, so load it.  
        ThisDrawing.Linetypes.Load "BORDER", "acad.lin"  
        ThisDrawing.ActiveLinetype = _  
            ThisDrawing.Linetypes.Item("BORDER")  
    End If  
    On Error GoTo 0  
    ' Turn off error trapping  
  
    ' Create a circle object in model space  
    Dim center(0 To 2) As Double  
    Dim radius As Double  
    Dim circleObj As AcadCircle  
    center(0) = 2  
    center(1) = 2  
    center(2) = 0  
    radius = 4  
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)  
    circleObj.Update  
    MsgBox ("Here is the circle with the original linetype")  
  
    ' Set the linetype scale of a circle to 3  
    circleObj.LinetypeScale = 3#  
    circleObj.Update  
    MsgBox ("Here is the circle with the new linetype")  
  
    ' Restore original active linetype  
    ThisDrawing.ActiveLinetype = currLinetype  
End Sub
```

Assigning Layers, Colors, and Linetypes to Objects

Once you've defined layers, colors, and linetypes, you can assign them to objects in your drawing. You can group associated components of a drawing by assigning objects to layers. You can control layer visibility, color, and linetype and specify whether objects on a layer can be edited. You can move objects from one layer to another and change the name of a layer.

The number of layers in a drawing and the number of objects per layer are virtually unlimited. You can assign a name to each layer and select any combination of layers for display.

You can define blocks from objects that were originally drawn on different layers with different colors and linetypes. You can preserve the layer, color, and linetype information of objects in a block. Then, each time you insert the block, you have each object drawn on its original layer with its original color and linetype.

Changing an Object's Layer

Once you have created an object and assigned layer, color, and linetype properties to it, you may wish to change the object's layer. Changing an object's layer is useful if you accidentally create an object on the wrong layer or decide to change your layer organization later.

To change an object's layer, use the [Layer](#) property provided for that object. The Layer property takes the name of the layer as input.

Moving an object to a different layer

This example creates a circle on the active layer and then creates a new layer called "ABC". It then moves the circle to the new layer.

```
Sub Ch4_MoveObjectNewLayer()  
    ' Create a circle  
    Dim circleObj As AcadCircle  
    Dim center(0 To 2) As Double  
    Dim radius As Double  
    center(0) = 2: center(1) = 2: center(2) = 0  
    radius = 1  
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)  
  
    ' Create a new layer called "ABC"  
    Dim layerObj As AcadLayer  
    Set layerObj = ThisDrawing.Layers.Add("ABC")  
  
    ' Assign the circle to the "ABC" layer  
    circleObj.Layer = "ABC"  
    circleObj.Update  
End Sub
```

Changing an Object's Color

To change an object's color, use the [Color](#) property provided for that object. You can assign colors to individual objects in a drawing. Each color is identified by an ACI number, an integer from 1 to 255. Standard color constants are available for colors 1 to 7. The color constants are `acRed`, `acYellow`, `acGreen`, `acBlue`, `acMagenta`, and `acWhite`.

Setting a color for the object overrides the color setting for the layer on which the object resides. If you want to retain an object on a specific layer but you don't want it to keep the color of that layer, you can change the object's color.

Changing the color of a circle

This example creates a circle and then colors the circle red.

```
Sub Ch4_ColorCircle()  
    ' Create a circle  
    Dim circleObj As AcadCircle  
    Dim center(0 To 2) As Double  
    Dim radius As Double  
    center(0) = 2: center(1) = 2: center(2) = 0  
    radius = 1  
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)  
  
    ' Color the circle red  
    circleObj.Color = acRed  
    circleObj.Update  
End Sub
```

Changing an Object's Linetype

By default, objects inherit the linetype of the layer on which they are created. To change an object's linetype, use the [Linetype](#) property provided for that object. The Linetype property takes the name of the linetype to assign to the object as input.

NOTE Before you can assign a linetype to an object, the linetype must be loaded into the current drawing. To load a linetype into the drawing, use the Load method.

For more information about linetypes, see “Overview of Lintypes” in chapter 14, “Control the Properties of Objects,” in the *User's Guide*.

Changing the linetype of a circle

This example creates a circle. It then attempts to load the linetype “CENTER” from the *acad.lin* file. If the linetype already exists, or the file does not exist, then a message is displayed. Finally, it sets the linetype for the circle to be “CENTER.”

```

Sub Ch4_ChangeCircleLinetype()
    On Error Resume Next

    ' Create a circle
    Dim circleObj As AcadCircle
    Dim center(0 To 2) As Double
    Dim radius As Double
    center(0) = 2: center(1) = 2: center(2) = 0
    radius = 1
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)

    Dim linetypeName As String
    linetypeName = "CENTER"

    ' Load "CENTER" line type from acad.lin file
    ThisDrawing.Linetypes.Load linetypeName, "acad.lin"
    If Err.Description <> "" Then MsgBox Err.Description

    ' Assign the circle the linetype "CENTER"
    circleObj.Linetype = "CENTER"
    circleObj.Update
End Sub

```

Saving and Restoring Layer Settings

You can save layer settings in a drawing and restore them later. This makes it easy to return to specified settings for all layers during different stages when completing a drawing or when plotting a drawing.

Layer settings include whether or not a layer is turned on, frozen, locked, plotted, and automatically frozen in new viewports, and the layer's color, linetype, linewidth, and plot style. You can specify which settings you want to save, and you can save different groups of settings for a drawing.

A special object, the `LayerStateManager`, provides functions for working with layer settings using `ActiveX`.

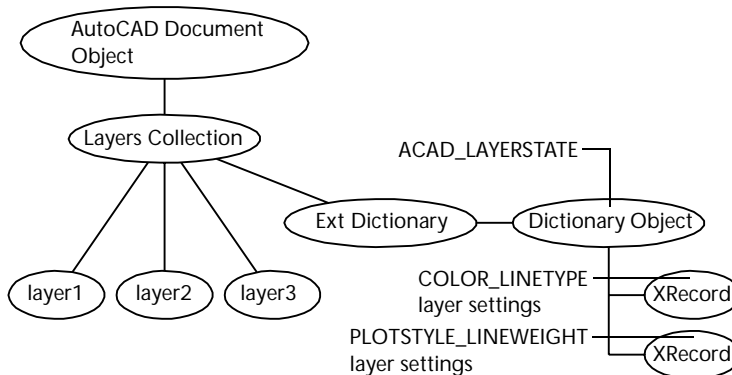
For more information about saving layer settings, see “Save and Restore Layer Settings” in chapter 14, “Control the Properties of Objects,” in the *User's Guide*.

Understanding How AutoCAD Saves Layer Settings

AutoCAD saves layer setting information in an extension dictionary in a drawing's Layers collection. When you first save layer settings in a drawing, AutoCAD does the following:

- Creates an extension dictionary in the Layers collection.
- Creates a Dictionary object named ACAD_LAYERSTATE in the extension dictionary.
- Stores the properties of each layer in the drawing in an XRecord object in the ACAD_LAYERSTATE dictionary. AutoCAD stores all layer settings in the XRecord, but identifies the specific settings you chose to save. When you restore the layer settings, AutoCAD restores only the settings you chose to save.

Each time you save another layer setting in the drawing, AutoCAD creates another XRecord object describing the saved settings and stores the XRecord in the ACAD_LAYERSTATE dictionary. The following diagram illustrates the process.



You do not need (and should not try) to interpret XRecords when working with layer settings using ActiveX. Use the functions of the LayerStateManager object to access saved layer settings.

Listing the saved layer settings in a drawing

If layer settings have been saved in the current drawing, the following code lists the names of all saved layer settings:

```
Sub Ch4_ListStates()  
    On Error Resume Next  
    Dim oLSMDict As AcadDictionary  
    Dim XRec As Object  
    Dim LayerstateNames As String  
  
    LayerstateNames = ""  
    ' Get the ACAD_LAYERSTATES dictionary, which is in the  
    ' extension dictionary in the Layers object.  
    Set oLSMDict = ThisDrawing.Layers._  
        GetExtensionDictionary.Item("ACAD_LAYERSTATES")  
    ' List the name of each saved layer setting. Settings are  
    ' stored as XRecords in the dictionary.  
    For Each XRec In oLSMDict  
        LayerstateNames = LayerstateNames + XRec.Name + vbCrLf  
    Next XRec  
    MsgBox "The saved layer settings in this drawing are: " + _  
        vbCrLf + LayerstateNames  
End Sub
```

Using the LayerStateManager to Manage Layer Settings

The LayerStateManager object is similar to the AutoCAD Utility object in that it provides a set of functions for manipulating data. These functions are methods for working with saved layer settings. Use the following LayerStateManager methods to work with saved layer settings:

Delete	Deletes a saved layer setting.
Export	Exports the specified saved layer setting to a file.
Import	Imports a saved layer setting from the specified file.
Rename	Renames a saved layer setting.
Restore	Restores the specified layer setting in the current drawing.
Save	Saves the specified layer states and properties.
SetDataBase	Associates an AutoCAD database with the LayerStateManager.

To access the **LayerStateManager** object, use the **GetInterfaceObject** method.

```
Dim oLSM As AcadLayerStateManager
Set oLSM = ThisDrawing.Application._
    GetInterfaceObject("AutoCAD.AcadLayerStateManager")
```

After you retrieve the **LayerStateManager** object, you must associate a database with it before you can access the object's methods. Use the **SetDatabase** method to associate a database with the **LayerStateManager**.

```
oLSM.SetDatabase ThisDrawing.Database
```

Saving Layer Settings

Use the **Save** method to save a set of layer settings in a drawing. The **Save** method accepts two parameters. The first parameter is a string naming the layer settings you are saving. The second parameter identifies the layer properties you want to save. Use the constants in the following table to identify layer properties.

Constants for layer properties	
Constant name	Layer property
acLsAll	All layer settings
acLsColor	Color
acLsFrozen	Frozen or thawed
acLsLineType	Linetype
acLsLineWeight	Lineweight
acLsLocked	Locked or unlocked
acLsNewViewport	New viewport layers frozen or thawed
acLsNone	None
acLsOn	On or off
acLsPlot	Plotting on or off
acLsPlotStyle	Plot style

Add the constants together to specify multiple properties.

If you try to save layer settings under a name that already exists, an error is returned. You must rename or delete the existing saved layer settings before you can reuse the name.

Saving a layer's color and linetype settings

The following code saves the color and linetype settings of the current layer under the name Col orLi netype.

```
Sub Ch4_SaveLayerCol orAndLi netype()  
    Dim oLSM As AcadLayerStateManager  
    ' Access the LayerStateManager object  
    Set oLSM = ThisDrawing.Application. _  
        GetInterfaceObject("AutoCAD.AcadLayerStateManager")  
    ' Associate the current drawing database with LayerStateManager  
    oLSM.SetDatabase ThisDrawing.Database  
    oLSM.Save "Col orLi netype", acLsCol or + acLsLi neType  
End Sub
```

Renaming a saved layer setting

The following code renames the Col orLi netype layer settings to Ol dCol orLi netype.

```
Sub Ch4_RenameLayerSettings()  
    Dim oLSM As AcadLayerStateManager  
    Set oLSM = ThisDrawing.Application. _  
        GetInterfaceObject("AutoCAD.AcadLayerStateManager")  
    oLSM.SetDatabase ThisDrawing.Database  
    oLSM.Rename "Col orLi netype", "Ol dCol orLi netype"  
End Sub
```

Deleting a saved layer setting

The following code deletes layer settings that were saved under the name Col orLi netype.

```
Sub Ch4_DeleteCol orAndLi netype()  
    Dim oLSM As AcadLayerStateManager  
    Set oLSM = ThisDrawing.Application. _  
        GetInterfaceObject("AutoCAD.AcadLayerStateManager")  
    oLSM.SetDatabase ThisDrawing.Database  
    oLSM.Delete "Col orLi netype"  
End Sub
```

Restoring Layer Settings

The Restore method resets all layer settings in the current drawing to values that were saved earlier. For example, if you save the drawing's color and linetype settings under the name "ColorLinetype" and subsequently change those settings, restoring "ColorLinetype" resets the layers to the colors and linetypes they had when "ColorLinetype" was saved. If you add new layers to the drawing after saving "ColorLinetype," those new layers are not affected when you restore "ColorLinetype."

Restoring the color and linetype settings of a drawing's layers

Assuming that the color and linetype settings of the layers in the current drawing were previously saved under the name "ColorLinetype," the following code resets the color and linetype settings of each layer in the drawing to the value they had when "ColorLinetype" was saved.

```
Sub Ch4_RestoreLayerSettings()  
    Dim oLSM As AcadLayerStateManager  
    Set oLSM = ThisDrawing.Application._  
        GetInterfaceObject("AutoCAD.AcadLayerStateManager")  
    oLSM.SetDatabase ThisDrawing.Database  
    oLSM.Restore "ColorLinetype"  
End Sub
```

Exporting and Importing Saved Layer Settings

You can export and import saved layer settings to use those settings in other drawings. Use the LayerStateManager's Export method to save layer settings to a file; use the Import method to import saved layer settings into a drawing.

NOTE Importing layer settings does not restore them; you must use the Restore method to set the layers in your drawing to the imported settings.

The Export method accepts two parameters. The first parameter is a string identifying the saved layer settings you are exporting. The second parameter is the name of the file you are exporting the settings to. If you do not specify a path for the file, it is saved in the AutoCAD installation directory. If the file name you specified already exists, the existing file is overwritten. Use a .las extension when naming files; this is the extension AutoCAD recognizes for exported layer setting files.

The Import method accepts one parameter: a string naming the file that contains the layer settings you are importing.

When you are importing layer settings, an error condition is raised if any properties referenced in the saved settings are not available in the drawing you're importing to. The import is completed, however, and default properties are used. For example, if an exported layer is set to a linetype that is not loaded in the drawing it is being imported into, an error condition is raised and the drawing's default linetype is substituted. Your code should account for this error condition and continue processing if it is raised.

If the imported file defines settings for layers that do not exist in the current drawing, those layers are created in the current drawing. When you use the `Restore` method, the properties specified when the settings were saved are assigned to the new layers; all other properties of the new layers are assigned default settings.

Exporting saved layer settings

The following code exports saved layer settings to a file named *Colortype.las*.

```
Sub Ch4_ExportLayerSettings()
    Dim oLSM As AcadLayerStateManager
    Set oLSM = ThisDrawing.Application. _
        GetInterfaceObject("AutoCAD.AcadLayerStateManager")
    oLSM.SetDatabase ThisDrawing.Database
    oLSM.Export "ColorLinetype", "c:\my documents\Col orLType.las"
End Sub
```

Importing saved layer settings

The following code imports layer settings from a file named *Colortype.las*.

```
Sub Ch4_ImportLayerSettings()
    Dim oLSM As AcadLayerStateManager
    Set oLSM = ThisDrawing.Application. _
        GetInterfaceObject("AutoCAD.AcadLayerStateManager")
    oLSM.SetDatabase ThisDrawing.Database

    ' If the drawing you're importing to does not contain
    ' all the linetypes referenced in the saved settings,
    ' an error is returned. The import is completed, though,
    ' and the default linetype is used.
    On Error Resume Next
    oLSM.Import "c:\my documents\Col orLType.las"
    If Err.Number = -2145386359 Then
        ' Error indicates a linetype is not defined
        MsgBox ("One or more linetypes specified in the imported " + _
            "settings is not defined in your drawing")
    End If
    On Error GoTo 0
End Sub
```

Adding Text to Drawings

Text conveys important information in your drawing. Use text for title blocks, to label parts of the drawing, to give specifications, or to make annotations.

AutoCAD provides various ways to create text. For short, simple entries, use line text. For longer entries with internal formatting, use multiline text (mtext). Although all entered text uses the current text style, which establishes the default font and format settings, you can use several methods to customize the text appearance.

For more information about working with text, see chapter 19, “Notes and Labels,” in the *User's Guide*.

Working with Text Styles

All text in an AutoCAD drawing has a style associated with it. When you enter text, AutoCAD uses the current text style, which sets the font, size, angle, orientation, and other text characteristics. You can use or modify the default style or create and load a new style. Once you've created a style, you can modify its attributes or delete it when you no longer need it.

For more information about text styles, see “Work with Text Styles” in chapter 19, “Notes and Labels,” in the *User's Guide*.

Creating and Modifying Text Styles

New text inherits height, width factor, obliquing angle, and text generation properties from the current text style. To create a text style, use the Add method to create a new TextStyle object and add it to the TextStyles collection. The Add method takes a TextStyle name as input. Once created, you cannot change the name of a text style through AutoCAD ActiveX Automation.

Style names can contain letters, numbers, and the special characters dollar sign (\$), underscore (_), and hyphen (-). AutoCAD converts the characters to uppercase. If you don't enter a style name, AutoCAD automatically names the style Style n , where n is a number that starts at 1. Each new style is shown in increments of 1.

You can modify an existing style by changing the properties of the `TextStyle` object. You can also update existing text of that style type to reflect the changes. Use the following properties to modify a `TextStyle` object:

<code>FontFile</code>	Specifies the file associated with a font (character style).
<code>BigFontFile</code>	Specifies the special shape definition file used for a non-ASCII character set, such as Kanji.
<code>Height</code>	Specifies the character height.
<code>Width</code>	Specifies the expansion or compression of the characters.
<code>ObliqueAngle</code>	Specifies the slant of the characters.
<code>TextGenerationFlag</code>	Specifies backward text, upside-down text, or both.

If you change an existing style's font or orientation, all text using that style is changed to use the new font or orientation. Changing text height, width factor, and oblique angle does not change existing text but does change subsequently created text objects.

NOTE You must call the `Regen` or `Update` methods to see any changes to the above properties.

For more information about creating text styles, see “Create and Modify Text Styles” in chapter 19, “Notes and Labels,” in the *User's Guide*.

Assigning Fonts

Fonts define the shapes of the text characters that make up each character set. A single font can be used by more than one style. To assign a font to a text style, use the `FontFile` property of the `TextStyle` object. By entering the font file containing an AutoCAD-compiled SHX font, you assign that font to the text style.

For more information about assigning fonts to text styles, see “Assign Text Fonts” in chapter 19, “Notes and Labels,” in the *User's Guide*.

Setting text fonts

This example gets the current font values for the active text style and then changes the typeface for the font to “PlayBill.” The new font is then set using the SetFont method. To see the effects of changing the typeface, add some MText or Text to your current drawing before running the example. Note that, if you don’t have the PlayBill font on your system, you need to substitute a font you do have in order for this example to work.

```
Sub Ch4_UpdateTextFont()  
  
    MsgBox ("Look at the text now...")  
  
    Dim typeFace As String  
    Dim SavetypeFace As String  
    Dim Bold As Boolean  
    Dim Italic As Boolean  
    Dim charSet As Long  
    Dim PitchandFamily As Long  
  
    ' Get the current settings to fill in the  
    ' default values for the SetFont method  
    ThisDrawing.ActiveTextStyle.GetFont typeFace, _  
        Bold, Italic, charSet, PitchandFamily  
  
    ' Change the typeface for the font  
    SavetypeFace = typeFace  
    typeFace = "PlayBill"  
    ThisDrawing.ActiveTextStyle.SetFont typeFace, _  
        Bold, Italic, charSet, PitchandFamily  
    ThisDrawing.Regen acActiveViewport  
    MsgBox ("Now see how it looks after changing the font...")  
  
    ' Restore the original typeface  
    ThisDrawing.ActiveTextStyle.SetFont SavetypeFace, _  
        Bold, Italic, charSet, PitchandFamily  
    ThisDrawing.Regen acActiveViewport  
End Sub
```

Using TrueType Fonts

TrueType® fonts always appear filled in your drawing, however, when you plot, the TEXTFILL system variable controls whether the fonts are filled. By default TEXTFILL is set to 1 to plot the filled-in fonts. When you export the drawing to PostScript® format with the Export method and print it on a PostScript device, the font is plotted as designed.

For more information about using TrueType fonts, see “Assign Text Fonts” in chapter 19, “Notes and Labels,” in the *User’s Guide*.

Using Unicode and Big Fonts

AutoCAD supports the Unicode character-encoding standard. A Unicode font can contain 65,535 characters, with shapes for many languages. All AutoCAD SHX shape fonts are now Unicode fonts.

The text files for some alphabets, such as kanji, contain thousands of non-ASCII characters. To accommodate such text, AutoCAD supports a special type of shape definition known as a Big Font file. You can set a style to use both regular and Big Font files. Specify normal fonts using the [FontFile](#) property. Specify Big Fonts using the [BigFontFile](#) property.

NOTE Font file names cannot contain commas.

AutoCAD provides ways to substitute one font for another or to specify a default font. For more information see “Substituting Fonts” on page 169.

For more information about using Unicode and Big Fonts, see “Assign Text Fonts” in chapter 19, “Notes and Labels,” in the *User’s Guide*.

Changing font files

This example changes the `FontFile` and `BigFontFile` properties. You need to replace the path information listed in this example with path and file names appropriate for your system.

```
Sub Ch4_ChangeFontFiles()  
  ThisDrawing.ActiveTextStyle.BigFontFile = _  
    "C:/AutoCAD/Fonts/bigfont.shx"  
  ThisDrawing.ActiveTextStyle.FontFile = _  
    "C:/AutoCAD/Fonts/italic.shx"  
End Sub
```

Setting Text Height

Text height determines the size in drawing units of the letters in the font you are using. The value usually represents the size of the uppercase letters, with the exception of TrueType fonts.

For TrueType fonts, the value specified for text height might not represent the height of uppercase letters. The height specified represents the height of a capital letter plus an accent area reserved for accent marks and other marks used in non-English languages. The relative portion of areas assigned to capital letters and accent characters is determined by the font designer at the time the font is designed, and, consequently, will vary from font to font.

In addition to the height of a capital letter and the ascent area that make up the height specified by the user, TrueType fonts have a descent area for portions of characters that extend below the text insertion line. Examples of such characters are y, j, p, g, and q.

You specify the text height using the [Height](#) property. This property accepts positive numbers only.

For more information about setting text height, see “Set Text Height” in chapter 19, “Notes and Labels,” in the *User’s Guide*.

Changing the height of a Text object

This example creates a line of text and then changes the height of the text.

```
Sub Ch4_ChangeTextHeight()  
    Dim textObj As AcadText  
    Dim textString As String  
    Dim insertionPoint(0 To 2) As Double  
    Dim height As Double  
  
    ' Define the text object  
    textString = "Hello, World."  
    insertionPoint(0) = 3  
    insertionPoint(1) = 3  
    insertionPoint(2) = 0  
    height = 0.5  
  
    ' Create the text object in model space  
    Set textObj = ThisDrawing.ModelSpace.AddText(textString, insertionPoint, height)  
  
    ' Change the value of the Height to 1  
    textObj.Height = 1  
    textObj.Update  
End Sub
```

Setting Obliquing Angle

The obliquing angle determines the forward or backward slant of the text. The angle represents the offset from its vertical axis (90 degrees). To set the obliquing angle, use the [ObliqueAngle](#) property. The obliquing angle must be provided in radians. A positive angle denotes a lean to the right, a negative value will have 2π added to it to convert it to its positive equivalent.

For more information about the oblique angle, see “Set Text Obliquing Angle” in chapter 19, “Notes and Labels,” in the *User’s Guide*.

Creating oblique text

This example creates a Text object then slants the text 45 degrees.

```
Sub Ch4_ObliqueText()  
    Dim textObj As AcadText  
    Dim textString As String  
    Dim insertionPoint(0 To 2) As Double  
    Dim height As Double  
  
    ' Define the text object  
    textString = "Hello, World."  
    insertionPoint(0) = 3  
    insertionPoint(1) = 3  
    insertionPoint(2) = 0  
    height = 0.5  
  
    ' Create the text object in model space  
    Set textObj = ThisDrawing.ModelSpace.AddText(textString, insertionPoint, height)  
  
    ' Change the value of the ObliqueAngle  
    ' to 45 degrees (.707 radians)  
    textObj.ObliqueAngle = 0.707  
    textObj.Update  
End Sub
```

Setting Text Generation Flag

The text generation flag specifies if the text is displayed backward or upside-down. To set the text generation flag, use the [TextGenerationFlag](#) property. To make the text display backward, enter `acTextFlagBackward` for this property. To make the text display upside-down, enter `acTextFlagUpsideDown` for this property. To make the text display both backward and upside-down, add the two constants together by entering `acTextFlagBackward+acTextFlagUpsideDown` for this property.

Displaying text backward

This example creates a line of text, then sets it to display backward using the `TextGenerationFlag` property.

```
Sub Ch4_ChangingTextGenerationFlag()  
    Dim textObj As AcadText  
    Dim textString As String  
    Dim insertionPoint(0 To 2) As Double  
    Dim height As Double  
  
    ' Create the text object  
    textString = "Hello, World."  
    insertionPoint(0) = 3  
    insertionPoint(1) = 3  
    insertionPoint(2) = 0  
    height = 0.5  
    Set textObj = ThisDrawing.ModelSpace.AddText(textString, insertionPoint, height)  
  
    ' Change the value of the TextGenerationFlag  
    textObj.TextGenerationFlag = acTextFlagBackward  
    textObj.Update  
End Sub
```

Using Line Text (Text)

The text you add to your drawings conveys a variety of information. It may be a complex specification, title block information, a label, or even part of the drawing. For shorter entries that do not require multiple fonts or lines, create line text using the Text object. Line text is more convenient for labels.

For more information about this topic, see “Create Single-Line Text” in chapter 19, “Notes and Labels,” in the *User's Guide*.

Creating Line Text

Each individual line of text is a distinct object when using line text. To create a line text object, use the `AddText` method. This method requires three values as input: the text string, the insertion point, and the height of the text.

The text string is the actual text to be displayed. Unicode, control code, and special characters are accepted. The insertion point is a variant array containing three doubles representing the 3D WCS coordinate in the drawing to place the text. The height of the text is a positive number representing the height of the uppercase text. Height is measured in the current units.

Creating Line Text

This example creates a line of text in model space, at the coordinate (2, 2, 0).

```
Sub Ch4_CreateText()  
    Dim textObj As AcadText  
    Dim textString As String  
    Dim insertionPoint(0 To 2) As Double  
    Dim height As Double  
  
    ' Create the text object  
    textString = "Hello, World."  
    insertionPoint(0) = 2  
    insertionPoint(1) = 2  
    insertionPoint(2) = 0  
    height = 0.5  
    Set textObj = ThisDrawing.ModelSpace.AddText(textString, insertionPoint, height)  
    textObj.Update  
End Sub
```

Formatting Line Text

A Text object is created using the active text style. You can change the formatting of the Text object by changing the text style associated with it, or by editing the properties of the Text object. You cannot apply formats to individual words and characters.

To change a text style associated with an individual Text object, set the `StyleName` property to a new text style. Once you have changed the text style, use the `Update` method for the Text object to see the changes in your drawing.

In addition to the standard editable properties for entities (color, layer, linetype, and so forth), other properties that you can change on a Text object include the following:

Alignment	Specifies the horizontal and vertical alignment for the text.
InsertionPoint	Specifies the insertion point for the text.
ObliqueAngle	Specifies the oblique angle of the individual text object.
Rotation	Specifies the rotation angle in radians for the text.

<code>ScaleFactor</code>	Specifies the scale factor for the text.
<code>TextAlignmentPoint</code>	Specifies the alignment point for the text.
<code>TextGenerationFlag</code>	Specifies whether the text is displayed backward, upside-down, or both simultaneously.
<code>TextString</code>	Specifies the actual text string displayed.

Once you have changed a property, use the Update method to see the changes in your drawing.

NOTE For a complete list of methods and properties, see the Text object documentation in the *AutoCAD ActiveX and VBA Reference*.

For more information about formatting line text, see “Work with Text Styles” in chapter 19, “Notes and Labels,” in the *User’s Guide*.

Aligning Line Text

You can justify line text with one of the horizontal and vertical alignment options shown in the following illustration. Left alignment is the default. To set the horizontal and vertical alignment options, use the Alignment property.

For more information about aligning line text, see “Align Single-Line Text” in chapter 19, “Notes and Labels,” in the *User’s Guide*.

Realigning text

This example creates a Text object and a Point object. The Point object is set to the text alignment point, and is changed to a red crosshair so that it is visible. The text alignment is changed and a message box is displayed so that the macro execution is halted. This allows you to see the impact of changing the text alignment.

```

Sub Ch4_TextAlignment()
    Dim textObj As AcadText
    Dim textString As String
    Dim insertionPoint(0 To 2) As Double
    Dim height As Double

    ' Define the new Text object
    textString = "Hello, World."
    insertionPoint(0) = 3
    insertionPoint(1) = 3
    insertionPoint(2) = 0
    height = 0.5

    ' Create the Text object in model space
    Set textObj = ThisDrawing.ModelSpace.AddText(textString, insertionPoint, height)

    ' Create a point over the text alignment point,
    ' so we can better visualize the alignment process
    Dim pointObj As AcadPoint
    Dim alignmentPoint(0 To 2) As Double
    alignmentPoint(0) = 3
    alignmentPoint(1) = 3
    alignmentPoint(2) = 0
    Set pointObj = ThisDrawing.ModelSpace.AddPoint(alignmentPoint)
    pointObj.Color = acRed

    ' Set the point style to crosshair
    ThisDrawing.SetVariable "PDMODE", 2

    ' Align the text to the Left
    textObj.Alignment = acAlignmentLeft
    ThisDrawing.Regen acActiveViewport
    MsgBox "The Text object is now aligned left"

    ' Align the text to the Center
    textObj.Alignment = acAlignmentCenter

    ' Align the text to the point (necessary for
    ' all but left aligned text.)
    textObj.TextAlignmentPoint = alignmentPoint

    ThisDrawing.Regen acActiveViewport
    MsgBox "The Text object is now centered"

    ' Align the text to the Right
    textObj.Alignment = acAlignmentRight
    ThisDrawing.Regen acActiveViewport
    MsgBox "The Text object is now aligned right"

End Sub

```

Changing Line Text

Like any other object, Text objects can be moved, rotated, erased, and copied. You also can mirror text. If you do not want the text to be reversed when you mirror it, you can set the MIRRTEXT system variable to 0.

The following list represents a few of the methods a Text object has for use in editing. For a complete list, see the Text object documentation in the *AutoCAD ActiveX and VBA Reference*.

ArrayPolar	Creates a polar array.
ArrayRectangular	Creates a rectangular.
Copy	Copies the Text object.
Erase	Erases the Text object.
Mirror	Mirrors the Text object.
Move	Moves the Text object.
Rotate	Rotates the Text object.

Using Multiline Text (Mtext)

For long, complex entries, create multiline text (mtext). Multiline text fits a specified width but can extend vertically to an indefinite length. You can format individual words or characters within the mtext.

For more information about this topic, see “Create Multiline Text” in chapter 19, “Notes and Labels,” in the *User’s Guide*.

Creating Multiline Text

You can create a multiline text object (MText object) by using the AddMText method. This method requires three values as input: the text string, the insertion point in the drawing to place the text, and the width of the text bounding box.

The text string is the actual text to be displayed. Unicode, control code, and special characters are accepted. The insertion point is a variant array containing three doubles representing the 3D WCS coordinate in the drawing to place the text. The width of the text is a positive number representing the width of the bounding box for the text. Width is measured in the current units.

Once the MText object is created, you can apply the text height, justification, rotation angle, and style to the MText object, or apply character formatting to selected characters.

Refer to the entry on MText in the *ActiveX and VBA Reference* for a list of methods and properties that apply to the MText object.

For more information about creating multiline text, see “Create Multiline Text” in chapter 19, “Notes and Labels,” in the *User’s Guide*.

Creating Multiline Text

The following code creates an MText object in model space, at the coordinate (2, 2, 0).

```
Sub Ch4_CreateMText()  
    Dim mtextObj As AcadMText  
    Dim insertPoint(0 To 2) As Double  
    Dim width As Double  
    Dim textString As String  
  
    insertPoint(0) = 2  
    insertPoint(1) = 2  
    insertPoint(2) = 0  
    width = 4  
    textString = "This is a text string for the mtext object."  
  
    ' Create a text Object in model space  
    Set mtextObj = ThisDrawing.ModelSpace.AddMText(insertPoint, width, textString)  
  
    ZoomAll  
End Sub
```

Formatting Multiline Text

New text automatically assumes the characteristics of the current text style. The STANDARD text style is the default. You can override the default text style by applying formatting to individual characters and applying properties to the Text object. You also can indicate formatting or special characters using the methods described in this section.

Orientation options such as style, justification, width, and rotation affect all text within the mtext text boundary, not specific words or characters. Use the [AttachmentPoint](#) property to change the justification of MText, and the [Rotation](#) property to control the angle of rotation of the text boundary.

The [StyleName](#) property sets the default fonts and formatting characteristics for new text. As you create text, you can select which style you want to use from a list of existing styles. When you change the style of an MText object that has character formatting applied to any portion of the text, the style is applied to the entire object, and some formatting of characters might not be

retained. For instance, changing from a TrueType style to a style using an SHX font or to another TrueType font causes the text to use the new font for the entire object, and any character formatting is lost.

Formatting options such as underlining, stacked text, or fonts can be applied to individual words or characters within a paragraph. You also can change color, font, and text height. You can change the spaces between text characters or increase the width of the characters. To apply formatting, use the ASCII equivalent of the format codes shown in the following table:

Mtext formatting codes			
Format code	Purpose	Enter this...	To produce this...
\O...\o	Turns overline on and off	Autodesk \OAutoCAD\o 2000	Autodesk <u>AutoCAD</u> 2000
\L...\l	Turns underline on and off	Autodesk \LAutoCAD\l 2000	Autodesk <u>AutoCAD</u> 2000
\~	Inserts a nonbreaking space	Autodesk AutoCAD\~2000	Autodesk AutoCAD 2000
\\	Inserts a backslash	Autodesk \\AutoCAD	Autodesk \AutoCAD
\{...\}	Inserts an opening and closing brace	Autodesk \{AutoCAD\} 2000	Autodesk {AutoCAD} 2000
\File name;	Changes to the specified font file	Autodesk \Ftimes; AutoCAD 2000	Autodesk AutoCAD
\Hvalue;	Changes to the text height specified in drawing units	Autodesk \H2;AutoCAD	Autodesk AutoCAD
\Hvaluex;	Changes the text height to a multiple of the current text height	Autodesk AutoCAD \H3x;2000	Autodesk AutoCAD 2000
\S...^...;	Stacks the subsequent text at the \, #, or ^ symbol	1.000\S+0.010^-0.000;	+0.010 1.000 -0.000

Mtext formatting codes (continued)

Format code	Purpose	Enter this...	To produce this...
<code>\Tvalue;</code>	Adjusts the space between characters, from .75 to 4 times	<code>\T2;Autodesk</code>	A u t o d e s k
<code>\Qangle;</code>	Changes obliquing angle	<code>\Q20;Autodesk</code>	<i>Autodesk</i>
<code>\Wvalue;</code>	Changes width factor to produce wide text	<code>\W2;Autodesk</code>	Autodesk
<code>\A</code>	Sets the alignment value; valid values: 0, 1, 2 (bottom, center, top)	<code>\A1;1\S1/2</code>	$1\frac{1}{2}$

Use curly braces ({}) to apply a format change only to the text within the braces. You can nest braces up to eight levels deep.

You also can enter the ASCII equivalent for control codes within lines or paragraphs to indicate formatting or special characters, such as tolerance or dimensioning symbols.

The following control characters can be used to create the text in the illustration. (For the ASCII equivalent of this string see the example below.)

{{\H1.5x; Big text} \A2; over text\A1;/\A0; under text}

Big text over text/
under text

For more information about formatting multiline text, see “Format Characters Within Multiline Text” in chapter 19, “Notes and Labels,” in the *User’s Guide*.

Using control characters to format text

This example creates and formats an MText object.

```
Sub Ch4_FormatMText()  
    Dim mtextObj As AcadMText  
    Dim insertPoint(0 To 2) As Double  
    Dim width As Double  
    Dim textString As String  
  
    insertPoint(0) = 2  
    insertPoint(1) = 2  
    insertPoint(2) = 0  
    width = 4  
  
    ' Define the ASCII characters for the control characters  
    Dim OB As Long ' Open Bracket {  
    Dim CB As Long ' Close Bracket }  
    Dim BS As Long ' Back Slash \  
    Dim FS As Long ' Forward Slash /  
    Dim SC As Long ' Semi colon ;  
    OB = Asc("{")  
    CB = Asc("}")  
    BS = Asc("\")  
    FS = Asc("/")  
    SC = Asc(";")  
  
    ' Assign the text string the following line of control  
    ' characters and text characters:  
    ' {{\H1.5x; Big text}\A2; over text\A1;\A0; under text}  
  
    textString = Chr(OB) + Chr(OB) + Chr(BS) + "H1.5x" _  
    + Chr(SC) + "Big text" + Chr(CB) + Chr(BS) + "A2" _  
    + Chr(SC) + "over text" + Chr(BS) + "A1" + Chr(SC) _  
    + Chr(FS) + Chr(BS) + "A0" + Chr(SC) + "under text" _  
    + Chr(CB)  
  
    ' Create a text Object in model space  
    Set mtextObj = ThisDrawing.ModelSpace.  
        AddMText(insertPoint, width, textString)  
    ZoomAll  
End Sub
```

Using Unicode Characters, Control Codes, and Special Characters

You can use Unicode characters, control codes, and special characters in your text string to represent symbols. (All non-text characters must be entered as their ASCII equivalent.)

You can create special characters by entering the following Unicode character strings:

Unicode character descriptions	
Unicode character	Description
\U+00B0	Degree symbol
\U+00B1	Plus/minus tolerance symbol
\U+2205	Diameter dimensioning symbol

In addition to using Unicode characters for special characters, you can specify a special character by including control information in the text string. Use a pair of percent signs (%%) to introduce each control sequence. For example, the following control code works with standard AutoCAD text and PostScript fonts to draw character number *nnn*:

```
%%nnn
```

In a VB or VBA text string, the example above would be entered as

```
Dim percent as Long
percent = ASC("%")
TextString = chr(percent) + chr(percent) + "nnn"
```

These control codes work with standard AutoCAD text fonts only:

Control code descriptions	
Control code	Description
%%o	Toggles overscore mode on and off
%%u	Toggles underscore mode on and off
%%d	Draws degree symbol
%%p	Draws plus and minus tolerance symbol
%%c	Draws diameter dimensioning symbol
%%%	Draws single percent sign

Substituting Fonts

You can designate fonts to be substituted for other fonts or as defaults when AutoCAD cannot find a font specified in a drawing.

The fonts used for the text in your drawing are determined by the text style and, for mtext, by individual font formats applied to sections of text.

You can use font mapping tables to ensure that your drawing uses only certain fonts, or to convert the fonts you used to other fonts. You can use these font mapping tables to enforce corporate font standards, or to facilitate offline printing. AutoCAD comes with a default font mapping table. You can edit this file using any ASCII text editor. You also can specify a different font mapping table file by using the [FontFileMap](#) property on the Preferences object.

For more information about font mapping tables and substituting fonts, see “Substitute Fonts” in chapter 19, “Notes and Labels,” in the *User’s Guide*.

Specifying an Alternative Default Font

If your drawing specifies a font that is not currently on your system, AutoCAD automatically substitutes the font designated as your alternate font. By default, AutoCAD uses the *simplex.shx* file. However, you can specify a different font if necessary. Use the [AltFontFile](#) property on the Preferences object to set the alternative font file name.

If you use a text style that uses a Big Font, you can map it to another font using the [AltFontFile](#) property. This system variable uses a default font file pair of *txt.shx*, *bigfont.shx*.

If AutoCAD cannot find a font file when a drawing is opened, it applies a default set of font substitution rules. For a description of the default rules, see “Specify an Alternative Font” in chapter 19, “Notes and Labels,” in the *User’s Guide*.

Checking Spelling

During a spelling check, AutoCAD matches the words in the drawing to the words in the current main dictionary. Any words you add are stored in the custom dictionary that is current at the time of the spelling check. For example, you can add proper names so that AutoCAD no longer identifies them as misspelled words.

To check spelling in another language, you can change to a different main dictionary.

There is no method for checking spelling provided in AutoCAD ActiveX Automation. However, you can specify a different main dictionary using the [MainDictionary](#) property, or a different custom dictionary using the [CustomDictionary](#) property on the Preferences object.

For more information about spellings checks, see “Check Spelling” in chapter 19, “Notes and Labels,” in the *User’s Guide*.

Dimensioning and Tolerancing

5

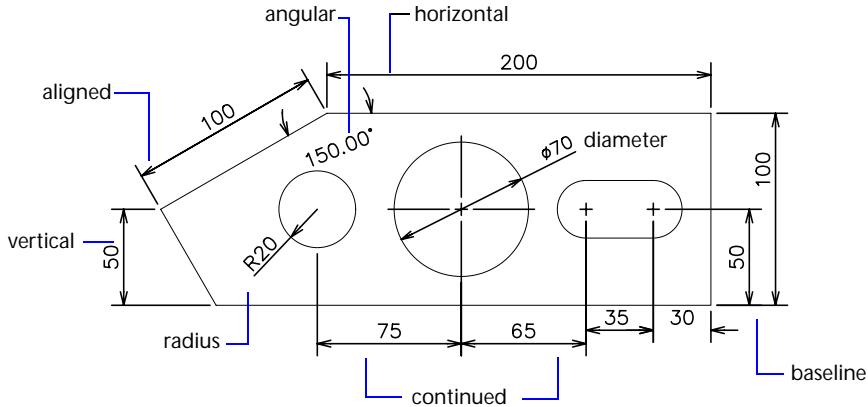
Dimensions add measurements to a drawing. Tolerances specify by how much a dimension can vary. With ActiveX Automation, dimensions can be managed with dimension styles and overrides.

In this chapter

- Reviewing Dimensioning Concepts
- Creating Dimensions
- Editing Dimensions
- Working with Dimension Styles
- Dimensioning in Model Space and Paper Space
- Creating Leaders and Annotation
- Using Geometric Tolerances

Reviewing Dimensioning Concepts

Dimensions show the geometric measurements of objects, the distances or angles between objects, or the *X* and *Y* coordinates of a feature. AutoCAD provides three basic types of dimensioning: linear, radial, and angular. Linear dimensions include aligned, rotated, and ordinate dimensions.



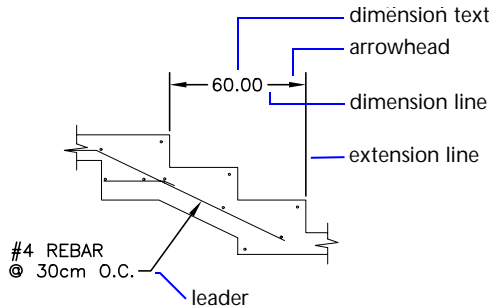
You can create dimensions for lines, multilines, arcs, circles, and polyline segments, or you can create dimensions that stand alone.

AutoCAD draws dimensions on the current layer. Every dimension has a dimension style associated with it, whether it's the default or one you define. The style controls characteristics such as color, text style, and linetype scale. Thickness information is not supported. Style families allow for subtle modifications to a base style for different types of dimensions. Overrides allow for style modifications to a specific dimension.

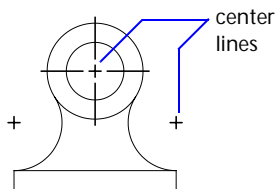
For more information about dimensions, see chapter 20, "Dimensions and Tolerances," in the *User's Guide*.

Looking at the Parts of a Dimension

This section briefly defines the parts of a dimension.



A dimension line is a line that indicates the direction and extent of a dimension. For an angular dimension, the dimension line is an arc. Extension lines, also called projection lines or witness lines, extend from the feature being dimensioned to the dimension line. Arrowheads, also called symbols of termination or just termination, are added to each end of the dimension line. Dimension text is a text string that usually indicates the actual measurement. The text may also include prefixes, suffixes, and tolerances. A leader is a solid line leading from some annotation to the referenced feature. A center mark is a small cross that marks the center of a circle or arc. Centerlines are broken lines that mark the center of a circle or arc.



See “Parts of a Dimension” in the *User’s Guide* for more information about the parts of a dimension.

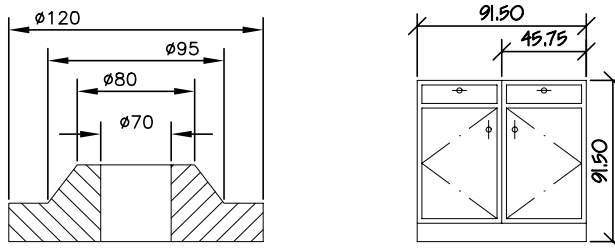
Defining the Dimension System Variables

The dimensioning system variables control the appearance of dimensions. The dimension system variables include: DIMAUNIT, DIMUPT, DIMTOFL, DIMFIT, DIMTIH, DIMTOH, DIMJUST, and DIMTAD. You can set these variables by using the SetVariable method. For example, the following line of code sets the DIMAUNIT system variable (the units format for angular dimensions) to radians (3):

See “Use Dimension Styles” in the *User’s Guide* for more information about the dimensioning system variables.

Setting Dimension Text Styles

Dimension text refers to any kind of text associated with dimensions, including measurements, tolerances (both lateral and geometric), prefixes, suffixes, and textual notes in single-line or paragraph form. You can use the default measurement computed by AutoCAD as the text, supply your own text, or suppress the text entirely. You can use dimension text to add information, such as special manufacturing procedures or assembly instructions.

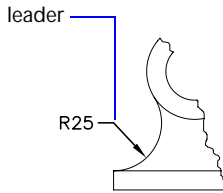


Single-line dimension text uses the active text style as specified by the `ActiveTextStyle` property. Paragraphs of text use the active text style with any modifications you make in your text string.

For more information about dimension text, see “Control Dimension Text” in the *User’s Guide*.

Understanding Leader Lines

A default leader line is a straight line with an arrowhead that refers to a feature in a drawing. Usually, a leader’s function is to connect annotation with the feature. Annotation in this case means paragraph text, blocks, or feature control frames. Such leader lines are different from the simple leader lines AutoCAD creates automatically for radial, diameter, and linear dimensions whose text won’t fit between extension lines.



Leader objects are associated with the annotation, so when the annotation is edited, the leader is updated accordingly. You can copy annotation used elsewhere in a drawing and append it to a leader, or you can create a new annotation. You can also create a leader with no annotation appended.

For more information about leaders, see “Overview of Creating Text and Leaders” in chapter 19, “Dimensions and Tolerances,” in the *User’s Guide*.

Understanding Associative Dimensions

Associative dimensions are dimensions in which all the lines, arrowheads, arcs, and text are drawn as a single-dimension object. The DIMASO system variable controls associative dimensioning and is on by default. If DIMASO is off, the dimension line, extension lines, arrowheads, leaders, and dimension text are drawn as separate objects. You can create a nonassociative dimension if you need to alter the dimension in ways that are not controlled by variables. In general, however, associative dimensions are easier to maintain because they can be modified as a single object.

To set or query a system variable, use the `SetVariable` and `GetVariable` methods.

Creating Dimensions

You can create linear, radial, angular, and ordinate dimensions.

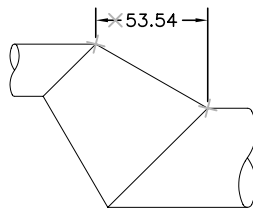
When creating dimensions, the active dimension style is used. Once created, you can modify the extension line origins, the dimension text location, and the dimension text content and its angle relative to the dimension line. You can also change the dimension style used by the dimension.

For more information about creating dimensions, see chapter 19, “Dimensions and Tolerances,” in the *User’s Guide*.

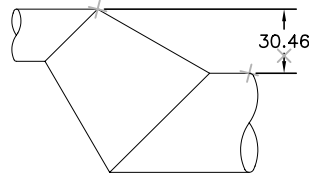
Creating Linear Dimensions

Linear dimensions can be aligned or rotated. Aligned dimensions have the dimension line parallel to the line along which the extension line origins lie. Rotated dimensions have the dimension line placed at an angle to the extension line origins.

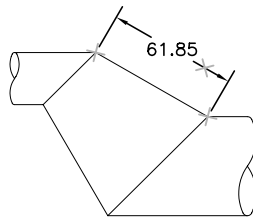
To create a linear dimension, use the `AddDimAligned` or `AddDimRotated` method. After you create linear dimensions, you can modify the text, the angle of the text, or the angle of the dimension line. In the following illustrations, the extension line origins are designated explicitly. The resulting dimension line location is also shown:



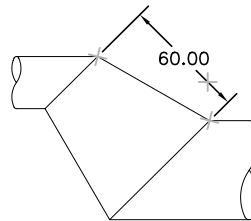
horizontal



vertical



aligned



rotated 315 degrees

To create an aligned dimension, use the `AddDimAligned` method. This method requires three coordinates as input: the origin of both extension lines and the text position.

To create a rotated dimension, use the `AddDimRotated` method. This method requires three coordinates and the angle of the dimension line as input. The three coordinates are the origin of both extension lines and the text position. The angle must be provided in radians and represents the angle of rotation for the dimension line.

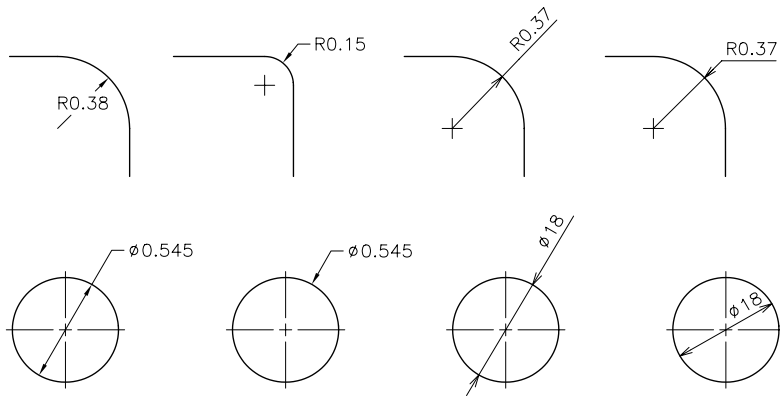
For additional information about creating linear dimensions, see “Create Linear Dimensions” in chapter 20, “Dimensions and Tolerances,” in the *User’s Guide*.

Creating Radial Dimensions

Radial dimensions measure the radii and diameters of arcs and circles. To create a radial dimension, use the `AddDimRadial` method.

Different types of radial dimensions are created depending on the size of the circle or arc, the `TextPosition` property, and the values in the `DIMUPT`, `DIMTOFL`, `DIMFIT`, `DIMTIH`, `DIMTOH`, `DIMJUST`, and `DIMTAD` dimension system variables. (System variables can be queried or set using the `GetVariable` and `SetVariable` methods.)

For horizontal dimension text, if the angle of the dimension line is more than 15 degrees from horizontal, and is outside the circle or arc, AutoCAD draws a hook line, also called a landing or dogleg. The hook line is one arrow-head long, and is placed next to the dimension text, as shown in the following illustrations:



To create radial dimensions, use the `AddDimRadial` or `AddDimDiametric` methods. These methods require three values as input: the coordinate of the circle or arc's center, the coordinate for the leader attachment, and the length of the leader.

These methods use the `LeaderLength` parameter as the distance from the `ChordPoint` to the point where the dimension will do a horizontal hook line to the annotation text (or stop if no hook line is necessary).

For additional information about creating radial dimensions, see “Create Radial Dimensions” in chapter 20, “Dimensions and Tolerances,” in the *User's Guide*.

Creating a radial dimension

This example creates a radial dimension in model space.

```
Sub Ch5_CreateRadialDimension()  
  Dim dimObj As AcadDimRadial  
  Dim center(0 To 2) As Double  
  Dim chordPoint(0 To 2) As Double  
  Dim leaderLen As Integer  
  
  ' Define the dimension  
  center(0) = 0  
  center(1) = 0  
  center(2) = 0  
  chordPoint(0) = 5  
  chordPoint(1) = 5  
  chordPoint(2) = 0  
  leaderLen = 5  
  
  ' Create the radial dimension in model space  
  Set dimObj = ThisDrawing.ModelSpace.AddDimRadial(center, chordPoint, leaderLen)  
  ZoomAll  
End Sub
```

NOTE The LeaderLength setting is only used during the creation of the dimension (and even then only if the dimension is set to use the default text position value). After the dimension is closed for the first time, changing the LeaderLength value will not affect how the dimension displays, but the new setting will be stored and will show up in DXF, LISP, and ADSRX applications.

Creating Angular Dimensions

Angular dimensions measure the angle between two lines or three points. For example, you can use them to measure the angle between two radii of a circle. The dimension line forms an arc.

To create an angular dimension, use the `AddDimAngular` method. This method requires three values as input: the angle vertex, the origins of the extension lines, and the text location. The `AngleVertex` is the center of the circle or arc, or the common vertex between the two lines being dimensioned. The origins of the extension lines are the points through which the two extension lines pass.

The `AngleVertex` can be the same as one of the origin points. If you need extension lines they will be added automatically.

For additional information about creating angular dimensions, see “Create Angular Dimensions” in chapter 20, “Dimensions and Tolerances,” in the *User’s Guide*.

Creating an angular dimension

This example creates an angular dimension in model space.

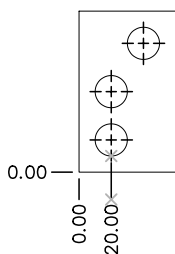
```
Sub Ch5_CreateAngularDimension()
    Dim dimObj As AcadDimAngular
    Dim angVert(0 To 2) As Double
    Dim FirstPoint(0 To 2) As Double
    Dim SecondPoint(0 To 2) As Double
    Dim TextPoint(0 To 2) As Double

    ' Define the dimension
    angVert(0) = 0
    angVert(1) = 5
    angVert(2) = 0
    FirstPoint(0) = 1
    FirstPoint(1) = 7
    FirstPoint(2) = 0
    SecondPoint(0) = 1
    SecondPoint(1) = 3
    SecondPoint(2) = 0
    TextPoint(0) = 3
    TextPoint(1) = 5
    TextPoint(2) = 0

    ' Create the angular dimension in model space
    Set dimObj = ThisDrawing.ModelSpace. _
        AddDimAngular(angVert, FirstPoint, SecondPoint, TextPoint)
    ZoomAll
End Sub
```

Creating Ordinate Dimensions

Ordinate, or datum, dimensions measure the perpendicular distance from an origin point, called the datum, to a dimensioned feature, such as a hole in a part. These dimensions prevent escalating errors by maintaining accurate offsets of the features from the datum.



Ordinate dimensions consist of an *X* or *Y* ordinate with a leader line. *X*-datum ordinate dimensions measure the distance of a feature from the datum along the *X* axis. *Y*-datum ordinate dimensions measure the same distance along the *Y* axis. AutoCAD uses the origin of the current UCS to determine the measured coordinates. The absolute value of the coordinate is used.

The text is aligned with the ordinate leader line regardless of the text orientation defined by the current dimension style. You can accept the default text or supply your own.

To create an ordinate dimension, use the `AddDimOrdinate` method. This method requires three values as input: a coordinate specifying the point to be dimensioned (A), a coordinate specifying the end of the leader (B), and a Boolean flag specifying whether the dimension is an *X*-datum ordinate dimension or a *Y*-datum ordinate dimension. If you enter `TRUE` for the Boolean flag, the method will create an *X*-datum ordinate dimension. If you enter `FALSE`, it will create a *Y*-datum ordinate dimension.

For additional information about creating ordinate dimensions, see “Create Ordinate Dimensions” in chapter 20, “Dimensions and Tolerances,” in the *User’s Guide*.

Creating an ordinate dimension

This example creates an ordinate dimension in model space.

```
Sub Ch5_CreatingOrdinateDimension()  
    Dim dimObj As AcadDimOrdinate  
    Dim definingPoint(0 To 2) As Double  
    Dim leaderEndPoint(0 To 2) As Double  
    Dim useXAxis As Long  
  
    ' Define the dimension  
    definingPoint(0) = 5  
    definingPoint(1) = 5  
    definingPoint(2) = 0  
    leaderEndPoint(0) = 10  
    leaderEndPoint(1) = 5  
    leaderEndPoint(2) = 0  
    useXAxis = 5  
  
    ' Create an ordinate dimension in model space  
    Set dimObj = ThisDrawing.ModelSpace.AddDimOrdinate(definingPoint, _  
        leaderEndPoint, useXAxis)  
    ZoomAll  
End Sub
```


Editing Dimensions

As with other graphical objects in AutoCAD, you can edit dimensions using the standard methods and properties provided for the object.

The following properties are available for most dimension objects:

Rotation	Specifies the rotation angle in radians for the dimension line.
StyleName	Specifies the name of the dimension style.
TextOverride	Specifies the text string for the dimension.
TextPosition	Specifies the dimension text position.
TextRotation	Specifies the rotation angle of the dimension text.
Measurement	Specifies the actual measurement for the dimension.

In addition, certain dimension objects provide properties for editing the extension line origins and leader length.

The following methods are included for dimension object editing:

ArrayPolar	Creates a polar array.
ArrayRectangular	Creates a rectangular array.
Copy	Copies the dimension object.
Erase	Erases the dimension object.
Mirror	Mirrors the dimension object.
Move	Moves the dimension object.
Rotate	Rotates the dimension object.
ScaleEntity	Scales the dimension object.

For more information about editing dimensions, see “Modify Existing Dimensions” in chapter 20, “Dimensions and Tolerances,” in the *User’s Guide*.

Overriding Dimension Text

The dimension value that is displayed can be replaced using the `TextOverride` property. Using this property you can completely replace the displayed value of the dimension, or you can append text to the value.

Modifying dimension text

This example appends some text to the value so that both the string and the dimension value are displayed.

```
Sub Ch5_Overri deDi mensi onText()  
    Dim di mObj As AcadDi mA ligned  
    Dim poi nt1(0 To 2) As Double  
    Dim poi nt2(0 To 2) As Double  
    Dim locati on(0 To 2) As Double  
  
    ' Define the dimensi on  
    poi nt1(0) = 5#: poi nt1(1) = 3#: poi nt1(2) = 0#  
    poi nt2(0) = 10#: poi nt2(1) = 3#: poi nt2(2) = 0#  
    locati on(0) = 7.5: locati on(1) = 5#: locati on(2) = 0#  
  
    ' Create an aligned dimension object in model space  
    Set di mObj = Thi sDrawi ng. Model Space. _  
        AddDi mA ligned(poi nt1, poi nt2, locati on)  
  
    ' Change the text string for the dimension  
    di mObj. TextOverride = "The value is <>"  
    di mObj. Update  
End Sub
```

Working with Dimension Styles

A named dimension style is a group of settings that determines the appearance of the dimension. Using named dimension styles, you can establish and enforce drafting standards for drawings.

All dimensions are created using the active dimension style. If you don't define or apply a style before creating dimensions, AutoCAD applies the default style, `STANDARD`. To set the active dimension style, use the `ActiveDimStyle` property.

To set up a parent dimension style, you begin by naming and saving a style. The new style is based on the current style and includes all subsequent changes to the layout of the dimension parts, the positioning of text, and the appearance of annotation. Annotation in this case means primary and alternate units, tolerances, and text.

For more information about dimension styles, see “Use Dimension Styles” in chapter 20, “Dimensions and Tolerances,” in the *User’s Guide*.

Creating, Modifying, and Copying Dimension Styles

To create a new dimension style, use the Add method. This method requires as input the name of the new dimension style.

AutoCAD ActiveX Automation allows you to add new dimension styles, and to change the active dimension style. You can also change the dimension style associated with a given dimension through the StyleName property.

You can also copy an existing style or set of overrides. Use the CopyFrom method to copy a dimension style from a source object to a newly dimension style. The source object can be another DimStyle object, a dimension, Tolerance, or Leader object, or even a Document object. If you copy the style settings from another dimension style, the style is duplicated exactly. If you copy the style settings from a dimension, Tolerance, or Leader object, the current settings, including any object overrides, are copied to the new style. If you copy the style of a Document object, the active dimension style, plus any drawing overrides, are copied to the new style.

Copying dimension styles and overrides

This example creates three new dimension styles and copies the current settings for the document, a given dimension style, and a given dimension to each new dimension style respectively. By following the appropriate setup before running this example, you will find that different dimension styles have been created.

- 1 Create a new drawing and make it the active drawing.
- 2 Create a linear dimension in the new drawing. This dimension should be the only object in the drawing.
- 3 Using the OPM, change the color of the dimension line to yellow.
- 4 Change the DIMCLRD system variable to 5 (blue).
- 5 Run the following example:

```

Sub Ch5_CopyDimStyles()
    Dim newStyle1 As AcadDimStyle
    Dim newStyle2 As AcadDimStyle
    Dim newStyle3 As AcadDimStyle

    Set newStyle1 = ThisDrawing.DimStyles.Add _
        ("Style 1 copied from a dim")
    Call newStyle1.CopyFrom(ThisDrawing.ModelSpace(0))

    Set newStyle2 = ThisDrawing.DimStyles.Add _
        ("Style 2 copied from Style 1")
    Call newStyle2.CopyFrom(ThisDrawing.DimStyles.Item _
        ("Style 1 copied from a dim"))

    Set newStyle3 = ThisDrawing.DimStyles.Add _
        ("Style 3 copied from the running drawing values")
    Call newStyle3.CopyFrom(ThisDrawing)
End Sub

```

Open the DIMSTYLE dialog. You should now have 3 dimension styles listed. Style 1 should have a yellow dimension line. Style 2 should be the same as Style 1. Style 3 should have a blue dimension line.

Overriding the Dimension Style

Each dimension has the capability of overriding settings in the dimension style for that dimension. The following properties are available for most dimension objects:

[AltRoundDistance](#)

Specifies the rounding of alternate units.

[AngleFormat](#)

Specifies the unit format for angular dimensions.

[Arrowhead1Block](#), [Arrowhead2Block](#)

Specifies the block to use as the custom arrowhead for the dimension line.

[Arrowhead1Type](#), [Arrowhead2Type](#)

Specifies the type of arrowhead for the dimension line.

[ArrowheadSize](#)

Specifies the size of dimension line arrowheads, leader line arrowheads, and hook lines.

[CenterMarkSize](#)

Specifies the size of the center mark for radial and diameter dimensions.

CenterType	Specifies the type of center mark for radial and diameter dimensions.
DecimalSeparator	Specifies the character to be used as the decimal separator in decimal dimension and tolerance values.
DimensionLineColor	Specifies the color of the dimension line for a dimension, leader, or tolerance object.
DimensionLineWeight	Specifies the line weight for the dimension lines.
DimLine1Suppress, DimLine2Suppress	Specifies the suppression of the dimension lines.
DimLineInside	Specifies the display of dimension lines inside the extension lines only.
ExtensionLineColor	Specifies the color for dimension extension lines.
ExtensionLineExtend	Specifies the distance the extension line extends beyond the dimension line.
ExtensionLineOffset	Specifies the distance the extension lines are offset from the origin points.
ExtensionLineWeight	Specifies the line weight for the extension lines.
ExtLine1EndPoint, ExtLine2EndPoint	Specifies the endpoint of extension lines.
ExtLine1StartPoint, ExtLine2StartPoint	Specifies the start point of extension lines.
ExtLine1Suppress, ExtLine2Suppress	Specifies the suppression of extension lines.

Fit	Specifies the placement of text and arrowheads inside or outside extension lines.
ForceLineInside	Specifies if a dimension line is drawn between the extension lines even when the text is placed outside the extension lines.
FractionFormat	Specifies the format of fractional values in dimensions and tolerances.
HorizontalTextPosition	Specifies the horizontal justification for dimension text.
LinearScaleFactor	Specifies a global scale factor for linear dimensioning measurements.
PrimaryUnitsPrecision	Specifies the number of decimal places displayed for the primary units of a dimension or tolerance.
SuppressLeadingZeros, SuppressTrailingZeros	Specifies the suppression of leading and trailing zeros in dimension values.
SuppressZeroFeet, SuppressZeroInches	Specifies the suppression of a zero foot and zero inch measurement in dimension values.
TextColor	Specifies the color of the text for dimension and tolerance objects.
TextGap	Specifies the distance between the dimension text and the dimension line when you break the dimension line to accommodate dimension text.
TextHeight	Specifies the height for the dimension or tolerance text.
TextInside	Specifies if the dimension text is to be drawn inside the extension lines.

TextInsideAlign	Specifies the position of dimension text inside the extension lines for all dimension types except ordinate.
TextMovement	Specifies how dimension text is drawn when text is moved.
TextOutsideAlign	Specifies the position of dimension text outside the extension lines for all dimension types except ordinate.
TextPosition	Specifies the dimension text position.
TextPrecision	Specifies the precision of angular dimension text.
TextPrefix	Specifies the dimension value prefix.
TextRotation	Specifies the rotation angle of the dimension text.
TextSuffix	Specifies the dimension value suffix.
ToleranceDisplay	Specifies if tolerances are displayed with the dimension text.
ToleranceHeightScale	Specifies a scale factor for the text height of tolerance values relative to the dimension text height.
ToleranceJustification	Specifies the vertical justification of tolerance values relative to the nominal dimension text.
ToleranceLowerLimit	Specifies the minimum tolerance limit for dimension text.
TolerancePrecision	Specifies the precision of tolerance values in primary dimensions.
ToleranceSuppressLeadingZeros	Specifies the suppression of leading zeros in tolerance values.

ToleranceSuppressTrailingZeros

Specifies the suppression of trailing zeros in dimension values.

ToleranceUpperLimit

Specifies the maximum tolerance limit for dimension text.

UnitsFormat

Specifies the unit format for all dimensions except angular.

VerticalTextPosition

Specifies the vertical position of text in relation to the dimension line.

Entering a user-defined suffix for an aligned dimension

This example creates an aligned dimension in model space and uses the TextSuffix property to allow the user to change the text suffix for the dimension.

```
Sub Ch5_AddTextSuffix()
    Dim dimObj As AcadDimAligned
    Dim point1(0 To 2) As Double
    Dim point2(0 To 2) As Double
    Dim location(0 To 2) As Double
    Dim suffix As String

    ' Define the dimension
    point1(0) = 0: point1(1) = 5: point1(2) = 0
    point2(0) = 5: point2(1) = 5: point2(2) = 0
    location(0) = 5: location(1) = 7: location(2) = 0

    ' Create an aligned dimension object in model space
    Set dimObj = ThisDrawing.ModelSpace.AddDimAligned(point1, point2, location)

    ThisDrawing.Application.ZoomAll

    ' Allow the user to change the text suffix for the dimension
    suffix = InputBox("Enter a new text suffix for the dimension" _
        , "Set Dimension Suffix", ": SUFFIX")

    ' Apply the change to the dimension
    dimObj.TextSuffix = suffix
    ThisDrawing.Regen acAllViewsports
End Sub
```


Dimensioning in Model Space and Paper Space

You can draw dimensions in both paper space and model space. However, if the geometry you're dimensioning is in model space, it's better to draw dimensions in model space, because AutoCAD places the definition points in the space where the geometry is drawn.

If you draw a dimension in paper space that describes geometry in your model, the paper space dimension does not change when you use editing commands or change the magnification of the display in the model space viewport. The location of the paper space dimensions also stays the same when you change a view from paper space to model space.

If you're dimensioning in paper space and the global scale factor for linear dimensioning (the DIMLFAC system variable) is set at less than 0, the distance measured is multiplied by the absolute value of DIMLFAC. If you're dimensioning in model space, the value of 1.0 is used even if DIMLFAC is less than 0. AutoCAD computes a value for DIMLFAC if you change the variable at the Dim prompt and select the Viewport option. AutoCAD calculates the scaling of model space to paper space and assigns the negative of this value to DIMLFAC.

Creating Leaders and Annotation

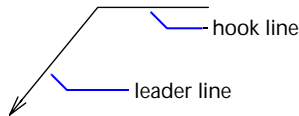
A leader is a line that connects some annotation to a feature in a drawing. Leaders and their annotation are associative, which means if you modify the annotation, the leader updates accordingly. Don't confuse the Leader object with the leader line AutoCAD automatically generates as part of a dimension line.

For more information about leaders, see "Create Text with Leaders" in chapter 19, "Notes and Labels," in the *User's Guide*.

Creating a Leader Line

You can create a leader line from any point or feature in a drawing and control its appearance as you draw it. Leaders can be straight line segments or smooth spline curves. Leader color is controlled by the current dimension line color. Leader scale is controlled by the overall dimension scale set in the active dimension style. The type and size of the arrowhead, if one is present, is controlled by the first arrowhead defined in the active style.

A small line known as a hook line usually connects the annotation to the leader. Hook lines appear with mtext and feature control frames if the last leader line segment is at an angle greater than 15 degrees from horizontal. The hook line is the length of a single arrowhead. If the leader has no annotation, it has no hook line.



To create a leader line, use the `AddLeader` method. This method requires three values as input: the array of coordinates to create the leader at, the annotation object (or `NULL` if the leader is to have no annotation), and the type of leader to create. The type of leader specifies whether the leader is to be a straight line or a smooth spline curve. It also determines whether or not the leader is to have arrows. Use one of the following constants to specify the type of leader: `acLineNoArrow`, `acLineWithArrow`, `acSplineNoArrow`, or `acSplineWithArrow`. These constants are mutually exclusive of each other.

Creating a leader line

This example creates a leader line in model space. There is no annotation associated with the leader line.

```
Sub Ch5_CreateLeader()  
    Dim leaderObj As AcadLeader  
    Dim points(0 To 8) As Double  
    Dim leaderType As Integer  
    Dim annotationObject As AcadObject  
  
    points(0) = 0: points(1) = 0: points(2) = 0  
    points(3) = 4: points(4) = 4: points(5) = 0  
    points(6) = 4: points(7) = 5: points(8) = 0  
    leaderType = acLineWithArrow  
    Set annotationObject = Nothing
```

```

' Create the leader object in model space
Set LeaderObj = ThisDrawing.ModelSpace. _
    AddLeader(points, annotationObject, LeaderType)
ZoomAll
End Sub

```

Adding the Annotation to a Leader Line

Leader annotations can be a Tolerance, MText, or BlockRef object. You can create a new annotation, or you can append a copy of an existing annotation. Annotation is added to the leader only when it is created.

To add an annotation when a leader is being created, input the annotation to the AddLeader method.

Leader Associativity

Leaders are associated with their annotation so that when the annotation moves, the endpoint of the leader moves with it. As you move text and feature control frame annotation, the final leader line segment alternates between attaching to the left side and to the right side of the annotation according to the relation of the annotation to the penultimate (second to last) point of the leader. If the midpoint of the annotation is to the right of the penultimate leader point, then the leader attaches to the right; otherwise, it attaches to the left.

Removing either object from the drawing using either the Erase, Add (to add a block), or WBlock method will break associativity. If the leader and its annotation are copied together in a single operation, the new copy is associative. If they are copied separately, they will not be associative. If associativity is broken for any reason, for example, by copying only the Leader object or by erasing the annotation, the hook line will be removed from the leader.

Associating a leader to the annotation

This example creates an MText object. A leader line is then created using the MText object as its annotation.

```

Sub Ch5_AddAnnotation()
    Dim LeaderObj As AcadLeader
    Dim mtextObj As AcadMText
    Dim points(0 To 8) As Double
    Dim insertionPoint(0 To 2) As Double
    Dim width As Double
    Dim LeaderType As Integer
    Dim annotationObject As Object
    Dim textString As String, msg As String

```

```

' Create the MText object in model space
textString = "Hello, World."
insertionPoint(0) = 5
insertionPoint(1) = 5
insertionPoint(2) = 0
width = 2
Set mtextObj = ThisDrawing.ModelSpace.AddMText(insertionPoint, width, textString)

' Data for Leader
points(0) = 0: points(1) = 0: points(2) = 0
points(3) = 4: points(4) = 4: points(5) = 0
points(6) = 4: points(7) = 5: points(8) = 0
leaderType = acLineWithArrow

' Create the Leader object in model space and associate
' the MText object with the leader
Set annotationObj = mtextObj
Set leaderObj = ThisDrawing.ModelSpace.AddLeader(points, annotationObj, leaderType)

ZoomAll
End Sub

```

Editing Leader Associativity

Except for the associativity relation between the leader and annotation, the leader and its annotation are entirely separate objects in your drawing. Editing of the leader does not affect the annotation, and editing of the annotation does not affect the leader.

Although text annotation is created using the DIMCLRT, DIMTXT, and DIMTXSTY system variables to define its color, height, and style, it cannot be changed by these system variables because it is not a true dimension object. Text annotation must be edited the same way as any other MText object.

Use the Evaluate method to evaluate the relation of the leader to its associated annotation. This method will update the leader geometry if necessary.

Editing Leaders

Any modifications to leader annotation that change its position affect the position of the endpoint of the associated leader. Also, rotating the annotation causes the leader hook line (if any) to rotate.

To resize a leader, you can scale it. Scaling updates only the scale of the selected object. For example, if you scale the leader, the annotation stays in the same position relative to the leader endpoint but isn't scaled.

In addition to scaling, you can also move, mirror, and rotate a leader. Use the ScaleEntity, Move, Mirror, and Rotate methods to edit the leader. You can

also change the text style associated with the annotation by using the Style-Name property.

For more information about editing leaders, see “Change Text with a Leader” in the *User's Guide*.

Using Geometric Tolerances

Geometric tolerancing shows deviations of form, profile, orientation, location, and runout of a feature. You add geometric tolerances in feature control frames. These frames contain all the tolerance information for a single dimension.

For more information about using feature control frames and working with geometric tolerances, see “Add Geometric Tolerances” in chapter 20, “Dimensions and Tolerances,” in the *User's Guide*.

Creating Geometric Tolerances

To create a geometric tolerance use the AddTolerance method. This method requires three values as input: the text string comprising the tolerance symbol, the location in the drawing to place the tolerance, and a directional vector specifying the direction of the tolerance. You can also copy, move, erase, scale, and rotate tolerances.

Creating a geometric tolerance

This example creates a simple geometric tolerance in model space.

```
Sub Ch5_CreateTolerance()  
    Dim toleranceObj As AcadTolerance  
    Dim textString As String  
    Dim insertionPoint(0 To 2) As Double  
    Dim direction(0 To 2) As Double  
  
    ' Define the tolerance object  
    textString = "Here is the Feature Control Frame"  
    insertionPoint(0) = 5  
    insertionPoint(1) = 5  
    insertionPoint(2) = 0  
    direction(0) = 1  
    direction(1) = 1  
    direction(2) = 0  
    ' Create the tolerance object in model space  
    Set toleranceObj = ThisDrawing.ModelSpace.  
        AddTolerance(textString, insertionPoint, direction)  
    ZoomAll  
End Sub
```

Editing Tolerances

Tolerances are influenced by several system variables: DIMCLRD controls the color of the feature control frame; DIMCLRT controls the color of the tolerance text; DIMGAP controls the gap between the feature control frame and the text; DIMTXT controls the size of the tolerance text; and DIMTXTSTY controls the style of the tolerance text. Use the SetVariable method to set the values of system variables.

Customizing Toolbars and Menus

6

AutoCAD ActiveX Automation gives you extensive control over the customization of menus and toolbars in the current AutoCAD session.

Using AutoCAD ActiveX/VBA, you can edit or augment the existing menu structure, or you can completely replace the current menu structure. You can also manipulate toolbars and right-click menus.

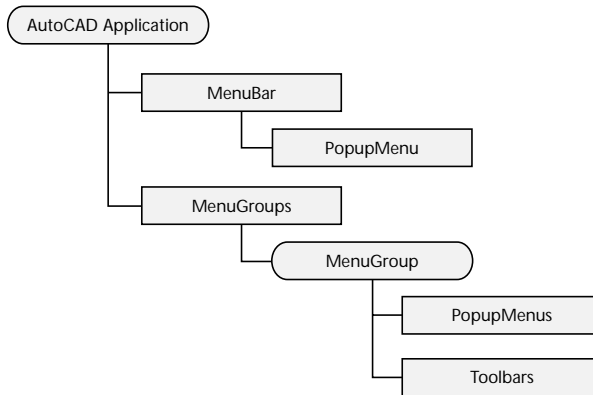
Menu customization can improve productivity by exposing application-specific tasks or by condensing tasks with multiple steps into a single menu selection.

In this chapter

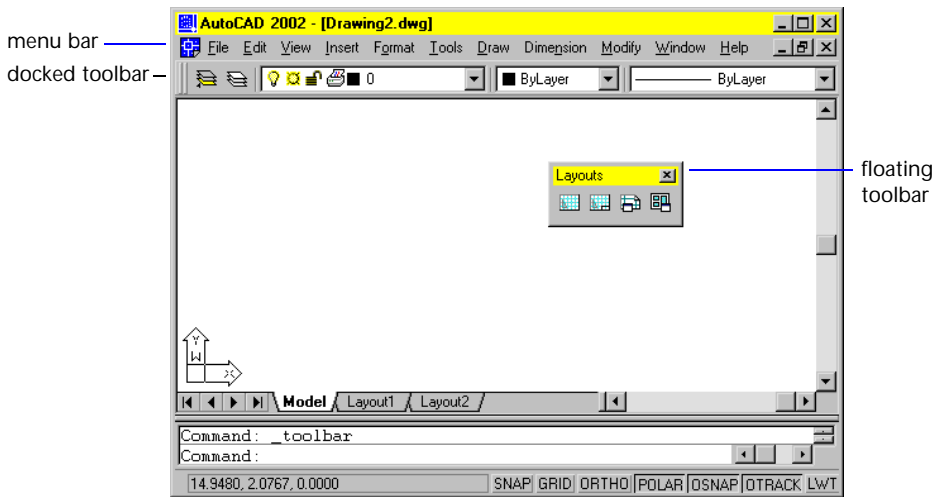
- Understanding the MenuBar and MenuGroups Collections
- Loading Menu Groups
- Creating New Menu Groups
- Changing the Menu Bar
- Creating and Editing Pull-Down and Shortcut Menus
- Creating and Editing Toolbars
- Creating Macros
- Creating Status-Line Help for Menu Items and Toolbar Items
- Adding Entries to the Right-Click Menu

Understanding the MenuBar and MenuGroups Collections

AutoCAD ActiveX provides several menu-related objects. The two most important are the MenuBar collection and the MenuGroups collection. The MenuBar collection contains all the menus that are displayed in the AutoCAD menu bar. The MenuGroups collection contains all the menu groups that are loaded in the current AutoCAD session. These menu groups contain all the menus that are available to the AutoCAD session, some or all of which may be displayed on the AutoCAD menu bar. In addition to the menus, the menu groups also contain all the toolbars that are available to the current AutoCAD session. Menu groups may also represent tile menus, screen menus, or tablet menus.

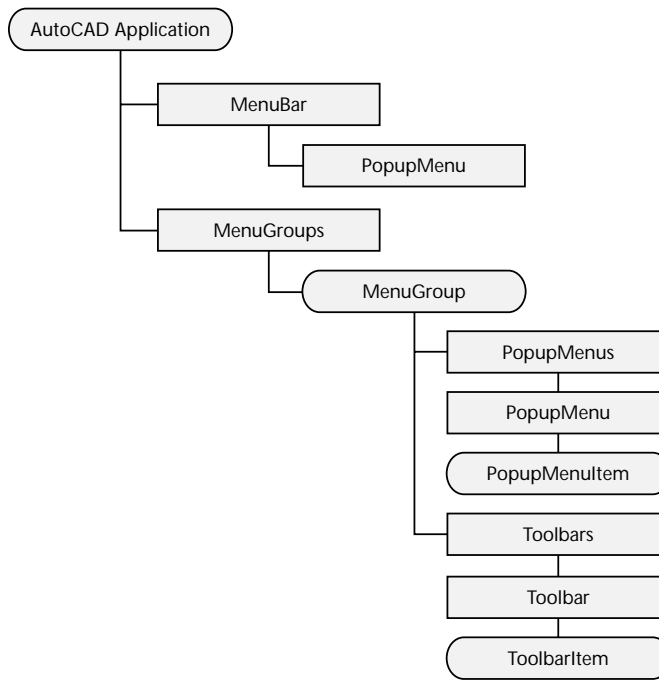


The MenuGroups collection contains the menu groups that are loaded in the current AutoCAD session. These menu groups contain all the menus that are available to the AutoCAD session, some or all of which may be displayed on the AutoCAD menu bar. In addition to the menus, the menu groups also contain all the toolbars that are available to the current AutoCAD session. Menu groups may also represent tile menus, screen menus, or tablet menus.



Each menu group contains a `PopupMenu` collection and a `Toolbars` collection. The `PopupMenu` collection contains all the menus within the menu group. Likewise, the `Toolbars` collection contains all the toolbars within the menu group.

Each `PopupMenu` is actually a collection that contains an individual object for each menu item that appears on that menu. Likewise, each `Toolbar` is also a collection that contains an individual object for each toolbar item that appears on that toolbar.



Loading Menu Groups

Menu groups are loaded into AutoCAD using the Load method. For example, the following code loads the menu file *acad.mnc*:

```
Thi sDrawing. Appl i cati on. MenuGroups. Load "acad. mnc"
```

When using the Load method, set the **BaseMenu** parameter to **TRUE** to load a new menu group to the menu bar. This will load the menu group as a base menu in the same manner as the **MENU** command in AutoCAD.

To load a new menu group as a partial menu, omit the **BaseMenu** parameter. This will load the menu group in the same manner as the **MENULOAD** command in AutoCAD. Once loaded into the MenuGroups collection, partial menus can be inserted into the menu bar by using the **InsertMenuInMenuBar** method, or the **InsertInMenuBar** method.

Once a menu group has been loaded, all the menus and toolbars defined by that menu group are available for use. You can

- Add new menus to the menu bar
- Remove menus from the menu bar
- Rearrange menus on the menu bar
- Add new items to an existing menu or toolbar
- Remove items from an existing menu or toolbar
- Create new menus and toolbars
- Float or dock toolbars
- Enable or disable menu and toolbar items
- Check or uncheck a menu item
- Change the tag, label, or help string of a menu or toolbar item
- Reassign the macros associated to a menu or toolbar item

NOTE You cannot edit image tile menu items, screen menus, or tablet menus using ActiveX Automation. However, you can load and unload these menu types using ActiveX Automation. For more information about these types of menus, see chapter 5, “Custom Menus” in the AutoCAD *Customization Guide*.

Creating New Menu Groups

AutoCAD ActiveX does not allow you to create new (empty) menu groups programmatically. However, you can load an existing menu group and save it out again with a new name and to a new menu file. For example, the following code saves the first menu group in the menu groups collection to a file called *MyMenu.mnc*:

```
ThisDrawing.Application.MenuGroups.Item(0). _  
SaveAs "MyMenu.mnc", acMenuFileCompiled
```

You can then edit the menu group to contain the exact configuration you desire. This process of creating new menu groups based on existing menu groups has the advantage of automatically providing you with basic menus such as File, Window, and Help.

Changing the Menu Bar

As you have already seen, the menu bar can be completely replaced by a new menu group if that group is loaded as the base menu. Additionally, individual menus on the menu bar can be added, removed, or rearranged.

Inserting Menus in the Menu Bar

To insert an existing menu to the menu bar, use the `InsertMenuInMenuBar` or the `InsertInMenuBar` method. Both methods accomplish the same goal—they insert an existing menu into the menu bar.

The difference between the two methods is the object from which they are called. The `InsertMenuInMenuBar` method is called from the `PopupMenu` collection. Using this method you can insert any menu from the collection into a specified location on the menu bar. This method requires as input the name of the menu to insert and the location on the menu bar to insert it.

The `InsertInMenuBar` method is called directly from the `PopupMenu` object to be inserted. The only input this method requires is a location on the menu bar. The name of the menu is not needed because you are calling the method directly from the object to be inserted.

You should use whichever method is more convenient for your application.

Inserting a menu in the menu bar

This example creates a new menu called `TestMenu` and inserts a menu item into it. The menu item is assigned the `OPEN` command. The menu is then displayed on the menu bar.

```
Sub Ch6_InsertMenu()  
    ' Define a variable for the current menu group  
    Dim currMenuGroup As AcadMenuGroup  
    Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)  
  
    ' Create a new menu  
    Dim newMenu As AcadPopupMenu  
    Set newMenu = currMenuGroup.Menus.Add("TestMenu")  
  
    ' Declare the variables for the menu item  
    Dim newMenuItem As AcadPopupMenuItem  
    Dim openMacro As String
```

```

' Assign the macro string the VB equivalent of
' "ESC ESC _open " and create the menu item
openMacro = Chr(vbKeyEscape) + Chr(vbKeyEscape) + "_open "
Set newMenuItem = newMenu.AddMenuItem(newMenu.Count + 1, _
    "Open", openMacro)

' Display the menu on the menu bar
currMenuGroup.Menus.InsertMenuItem newMenu, ""
End Sub

```

Removing Menus from the Menu Bar

To remove a menu from the menu bar, use the `RemoveMenuFromMenuBar` or the `RemoveFromMenuBar` method. Both methods accomplish the same goal—they remove a menu from the menu bar.

The difference between the two methods is the object from which they are called. The `RemoveMenuFromMenuBar` method is called from the `PopupMenu` collection. This method requires as input the name of the menu to remove, or the location on the menu bar of the menu to remove. For example, the following statement removes the menu added in “Inserting a menu in the menu bar”:

```
currMenuGroup.Menus.RemoveMenuFromMenuBar ("TestMenu")
```

The `RemoveFromMenuBar` method is called directly from the `PopupMenu` object to be removed. This method does not require any input. The name of the menu is not needed because you are calling the method directly from the object to be removed.

You should use whichever method is more convenient for your application.

NOTE Menus that have been removed from the menu bar are still available in their designated menu group. They are simply no longer visible to the user.

Rearranging Menu Items on the Menu Bar

To rearrange menus on the menu bar, insert and remove menus until the desired configuration is achieved.

Moving the first menu to the end of the menu bar

This example removes the first menu on the menu bar and inserts it as the last menu on the menu bar.

```
Sub Ch6_MoveMenu()  
    ' Define a variable to hold the menu to be moved  
    Dim moveMenu As AcadPopupMenu  
    Dim MyMenuBar As AcadMenuBar  
    Set MyMenuBar = ThisDrawing.Application.MenuBar  
  
    ' Set moveMenu equal to the first menu displayed  
    ' on the menu bar  
    Set moveMenu = MyMenuBar.Item(0)  
  
    ' Remove the first menu from the menu bar  
    MyMenuBar.Item(0).RemoveFromMenuBar  
  
    ' Add the menu back into the menu bar  
    ' in the last position on the bar  
    moveMenu.InsertInMenuBar(MyMenuBar.Count)  
End Sub
```

Creating and Editing Pull-Down and Shortcut Menus

AutoCAD ActiveX/VBA has the ability to customize two types of AutoCAD menus: pull-down menus and shortcut menus (sometimes called cursor menus). Both pull-down and shortcut menus are displayed as cascading menus. The shortcut menu can provide quick access to frequently used menu items such as Object Snap modes.

A pull-down menu can contain up to 999 menu items. A shortcut menu can contain up to 499 menu items. Both limits include all menus in the hierarchy. If the number of menu items in a menu exceeds these limits, AutoCAD ignores the extra items. If a pull-down or shortcut menu is taller than the available space on the graphics screen, it is truncated to fit on the screen.

Pull-down menus are always pulled down from the menu bar, but the shortcut menu is always displayed at or near the crosshairs on the graphics screen. The handling for both menu types is the same except the shortcut menu caption isn't included on the menu bar. The shortcut menu caption is not displayed at all. Access to the shortcut menu is through a single menu in the base menu group. The shortcut menu can be identified with the `ShortcutMenu` property. If the `ShortcutMenu` property returns `TRUE`, then the queried menu is the shortcut menu for the group.

For more information about working with menus, see “Pull-Down and Shortcut Menus” in chapter 5, “Custom Menus,” in the *Customization Guide*.

Creating New Menus

To create a new menu, use the `Add` method to add a new `PopupMenu` object to the `PopupMenu` collection.

To create a new shortcut menu, you must delete an existing shortcut menu. There can be only one shortcut menu per menu group. If there is no other shortcut menu in a menu group, you can add a menu with the label “POP0”. This will tell AutoCAD you want to create a shortcut menu.

The `Add` method requires as input the name (label) of the menu to add. This name becomes the title for the menu when it is loaded on the menu bar. The name is also the easiest way of identifying the menu within the collection.

The menu name can be a simple string or it can contain special codes. For a complete list of special codes, see “Summary of Pull-Down and Shortcut Menu Label Syntax” in chapter 5, “Custom Menus,” in the AutoCAD *Customization Guide*.

You can change the name of a menu once it has been created. To change the name of an existing menu, use the `Name` property for that menu.

Creating a new popup menu

This example creates a new popup menu called “TestMenu” in the first menu group of the `MenuGroups` collection.

```
Sub Ch6_CreateMenu()  
    Dim currMenuGroup As AcadMenuGroup  
    Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)  
  
    ' Create the new menu  
    Dim newMenu As AcadPopupMenu  
    Set newMenu = currMenuGroup.Menus.Add("TestMenu")  
End Sub
```

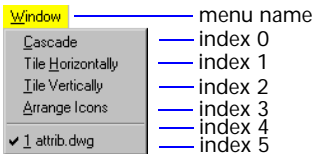
Adding New Menu Items to a Menu

To add a new menu item to a menu use the `AddMenuItem` method. This method creates a new `PopupMenuItem` object and adds it to the designated menu.

The `AddMenuItem` method takes four parameters as input: `Index`, `Label`, `Tag`, and `Macro`.

Specifying the Index Parameter

The `Index` parameter is an integer that specifies the position of the new menu item within the menu. The index begins with position zero (0) as the first position on the menu after the title. To add the new menu item to the end of a menu, set the `Index` parameter equal to the `Count` property of the menu. (The `Count` property of the menu represents the total number of menu items on that menu.)



In the diagram you will notice the first index position is zero (0) and the separators are listed as individual menu items with their own index position.

The `Count` property for the menu pictured would be six (6). To add a menu item between `Tile Horizontally` and `Tile Vertically`, set the `Index` parameter to two (2), which is the index of the `Tile Vertically` menu item. This inserts your new menu item into index two (2) and bumps all the remaining menu items down one index position.

Once a menu item has been created, you cannot change the index of the menu item through the `Index` property. To change the index of an existing menu item you must delete and re-add the menu item to a different position, or add or delete surrounding menu items until a proper placement is achieved.

Specifying the Label Parameter

A label is a string that defines the content and formatting of menu items. Menu item labels can contain DIESEL string expressions that conditionally alter the labels each time they are displayed. For more information about using DIESEL string expressions, see “DIESEL Expressions in Menus” in chapter 7, “DIESEL—String Expression Language,” in the *Customization Guide*.

In addition to the DIESEL string expressions, the label may contain special codes. For example, an ampersand (&) placed directly before a character specifies that character as the accelerator key. For a complete list of special codes, see “Summary of Pull-Down and Shortcut Menu Label Syntax” in chapter 5, “Custom Menus,” in the *Customization Guide*.

The text the user sees displayed for the menu item is called the Caption, and it is derived from the label by interpreting all the DIESEL string expressions and special codes contained in the label. For example, the label “&Edit” produces the caption “Edit.”

Once a menu item has been created, you can change the label for the menu item using the Label property.

Specifying the Tag Parameter

The tag, or name tag, is a string consisting of alphanumeric and underscore (_) characters. This string uniquely identifies the menu item within a given menu.

Once a menu item has been created, you can change the tag for the menu item using the TagString property.

Specifying the Macro Parameter

A macro is a series of commands that executes specific actions when a menu item is selected. Menu macros can simply be recordings of keystrokes that accomplish a task, or they can be a complex combination of commands, AutoLISP, DIESEL, or ActiveX programming code. For more information about menu macros, see “Menu Macros” in chapter 5, “Custom Menus,” in the *Customization Guide*.

Once a menu item has been created, you can change the macro for the menu item using the Macro property.

Adding menu items to a popup menu

This example creates a new menu called “TestMenu” and inserts a menu item. The menu item is given the name “Open,” and the macro assigned to the menu item is the OPEN command.

```

Sub Ch6_AddMenuItem()
    Dim currMenuGroup As AcadMenuGroup
    Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)

    ' Create the new menu
    Dim newMenu As AcadPopupMenu
    Set newMenu = currMenuGroup.Menus.Add("TestMenu")

    ' Add a menu item to the new menu
    Dim newItem As AcadPopupMenu.Item
    Dim openMacro As String
    ' Assign the macro the VBA equivalent of "ESC ESC _open "
    openMacro = Chr(vbKeyEscape) + Chr(vbKeyEscape) + "_open "

    Set newItem = newMenu.AddMenuItem _
        (newMenu.count + 1, "Open", openMacro)

    ' Display the menu on the menu bar
    newMenu.InsertInMenuBar _
        (ThisDrawing.Application.menuBar.count + 1)
End Sub

```

Adding Separators to a Menu

To add a separator to a menu use the `AddSeparator` method. This method creates a new `PopupMenu.Item` object and adds it to the designated menu. This kind of `PopupMenu.Item` object is assigned the type of `acSeparator`. The type of a menu item can be found through the `Type` property.

The `AddSeparator` method takes the `Index` parameter as its only input. The `Index` parameter is an integer that specifies the position of the separator within the menu. The index begins with position zero (0) as the first position on the menu after the title.

See “Enabling and disabling menu items” on page 212 for an example on adding separators to a menu.

Assigning an Accelerator Key to a Menu Item

To assign the accelerator key for a menu item through AutoCAD ActiveX/VBA, use the `Label` property of the given menu item. To specify an accelerator key, insert the ASCII equivalent of an ampersand (&) in the label directly in front of the character to be used as the accelerator. For example, the label `Chr(Asc("&")) + "Edit"` will be displayed as “Edit,” with the character “E” being used as the accelerator key.

There are other methods of specifying accelerator keys for AutoCAD menus and commands that are not currently available in AutoCAD ActiveX/VBA.

For more information about these methods, see “Accelerator Keys” in chapter 5, “Custom Menus,” in the *Customization Guide*.

Adding accelerator keys to menus

This example repeats the example from “Adding menu items to a popup menu” on page 205, adding accelerator keys for both the “TestMenu” and “Open” menus. The “s” is used as the accelerator key for the “TestMenu” menu and the “O” is used as the accelerator key for the “Open” menu.

```
Sub Ch6_AddAMenuItem()  
    Dim currMenuGroup As AcadMenuGroup  
    Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)  
  
    ' Create the new menu  
    Dim newMenu As AcadPopupMenu  
    Set newMenu = currMenuGroup.Menus.Add _  
        ("Te" + Chr(Asc("&"))) + "stMenu")  
  
    ' Add a menu item to the new menu  
    Dim newItem As AcadPopupMenuitem  
    Dim openMacro As String  
    ' Assign the macro the VBA equivalent of "ESC ESC _open "  
    openMacro = Chr(vbKeyEscape) + Chr(vbKeyEscape) + "_open "  
  
    Set newItem = newMenu.AddItem _  
        (newMenu.count + 1, Chr(Asc("&"))) _  
        + "Open", openMacro)  
  
    ' Display the menu on the menu bar  
    newMenu.InsertInMenuBar _  
        (ThisDrawing.Application.menuBar.count + 1)  
End Sub
```

Creating Cascading Submenus

To add a cascading submenu, create a submenu using the `AddSubmenu` method. This method creates a new `PopupMenuitem` object and adds it to the designated menu. This special kind of `PopupMenuitem` object is assigned the type of `acSubmenu`.

The `AddSubmenu` method takes three parameters as input: `Index`, `Label`, and `Tag`.

The `Index` parameter is an integer that specifies the position of the new menu item within the menu. The index begins with position zero (0) as the first position on the menu after the title. To add the new menu item to the end of a menu, set the `Index` parameter equal to the `Count` property of the menu. (The `Count` property of the menu represents the total number of menu items on that menu.)

The `Label` parameter is a string that defines the content and formatting of menu items. The text that the user sees displayed for the menu item is called the `Caption`, and it is derived from the label by interpreting all the DIESEL string expressions and special codes contained in the label. For example, the label “&Edit” produces the caption “Edit.”

The `Tag` parameter, or name tag, is a string consisting of alphanumeric and underscore (`_`) characters. This string uniquely identifies the menu item within a given menu.

The `AddSubmenu` method does not return the `PopupMenuItem` object that it creates. Instead, it returns the new menu that the submenu points to. The new menu, which is returned as a `PopupMenu` object, can then be populated as a normal menu would be. For information on populating a menu, see “Adding New Menu Items to a Menu” on page 204.

Creating and populating a submenu

This example creates a new menu called “TestMenu” and adds it to a submenu called “OpenFile.” The submenu is then populated with a menu item called “Open,” which opens a drawing when executed. Finally, the menu is displayed on the menu bar.

```
Sub Ch6_AddASubMenu()
    Dim currMenuGroup As AcadMenuGroup
    Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)

    ' Create the new menu
    Dim newMenu As AcadPopupMenu
    Set newMenu = currMenuGroup.Menus.Add("TestMenu")

    ' Add the submenu
    Dim FileSubMenu As AcadPopupMenu
    Set FileSubMenu = newMenu.AddSubMenu("", "OpenFile")

    ' Add a menu item to the sub menu
    Dim newMenuItem As AcadPopupMenuItem
    Dim openMacro As String
    ' Assign the macro the VB equivalent of "ESC ESC _open "
    openMacro = Chr(vbKeyEscape) + Chr(vbKeyEscape) + "_open "

    Set newMenuItem = FileSubMenu.AddMenuItem _
        (newMenu.count + 1, "Open", openMacro)

    ' Display the menu on the menu bar
    newMenu.InsertInMenuBar _
        (ThisDrawing.Application.menuBar.count + 1)
End Sub
```

Deleting Menu Items from a Menu

To remove menu items from a menu, use the Delete method found on the menu item.

WARNING! If you delete a menu item, do not call another method or property that would directly or indirectly cause the same menu file to be loaded again within the same macro. For example, after deleting a menu item, do not use the MenuGroup.Load method or the Preferences.Profiles.ActiveProfile property, or issue a "Menuload" command using the Document.SendCommand method. These items directly or indirectly cause the loading of menu files. You should only use these methods or properties in a separate macro.

Deleting a menu item from a menu

This example adds a menu item to the end of the last menu displayed on the menu bar. It then deletes the menu item.

```
Sub Ch6_DeleteMenuItem()  
    Dim LastMenu As AcadPopupMenu  
    Set LastMenu = ThisDrawing.Application.MenuBar._  
        Item(ThisDrawing.Application.MenuBar.Count - 1)  
  
    ' Add a menu item  
    Dim newMenuItem As AcadPopupMenu  
    Dim openMacro As String  
    ' Assign the macro the VB equivalent of "ESC ESC _open "  
    openMacro = Chr(vbKeyEscape) + Chr(vbKeyEscape) + "_open "  
  
    Set newMenuItem = LastMenu.AddMenuItem _  
        (LastMenu.Count + 1, "Open", openMacro)  
  
    ' Remove the menu item from the menu  
    newMenuItem.Delete  
End Sub
```

Exploring the Properties of Menu Items

All menu items share the following properties:

TagString

A tag, or name tag, is a string consisting of alphanumeric and underscore (_) characters. This string uniquely identifies the menu item within a given menu. Tags identify the accelerator keys (keyboard key sequences) that correspond to the menu item.

You can read or write the value of a tag by using the TagString property.

Label

A label is a string that defines the content and formatting of menu items.

Menu item labels can contain DIESEL string expressions that conditionally alter the labels each time they are displayed. For more information about using DIESEL string expressions, see “DIESEL Expressions in Menus” in chapter 7, “DIESEL—String Expression Language,” in the *Customization Guide*.

You can read or write the value of a label by using the Label property.

Caption

A caption is the text that the user sees displayed on the menu. This property is read-only and is derived from the Label property by removing any DIESEL string expressions.

You can read the value of a caption by using the Caption property.

Macro

A macro is a series of commands that executes specific actions when a menu item is selected. Menu macros can simply be recordings of keystrokes that accomplish a task, or they can be a complex combination of commands, AutoLISP, DIESEL, or ActiveX programming code. For more information about menu macros, see “Menu Macros” in chapter 5, “Custom Menus,” in the *Customization Guide*.

You can read or write the value of a menu macro by using the Macro property.

HelpString

A help string is the text string that appears in the AutoCAD status line when a user highlights a menu item for selection.

You can read or write the value of a help string by using the HelpString property.

Enable	<p>Using the Enable property you can enable or disable a menu item. You can also read the Enable property to determine if a menu item is currently enabled or disabled. Using this property to enable or disable a menu item overrides any setting for enabling in the DIESEL expression of the menu item.</p> <p>See “Enabling and disabling menu items” on page 212 for an example of disabling menu items.</p>
Check	<p>Using the Check property you can check or uncheck a menu item. You can also read the Check property to determine if a menu item is currently checked or unchecked. Using this property to check or uncheck a menu item overrides any setting for checking in the DIESEL expression of the menu item.</p>
Index	<p>The index of a menu item specifies the position of that menu item on the menu on which it belongs. The index position of a menu always begins with position 0. For example, if the item is the first item on a menu, it returns an index position of 0. If it is the second item on a menu, it returns an index position of 1 and so on.</p>
Type	<p>You can determine the type of a menu item by using the Type property. A menu item can be one of the following types: a regular menu, a separator, or the heading for a submenu. If the item is a regular menu item, this property returns <code>acMenuItem</code>. If the item is a separator, this property returns <code>acMenuSeparator</code>. If the item is a heading for a submenu, this property returns <code>acSubMenu</code>.</p>
SubMenu	<p>You can find the submenu by using the SubMenu property. If the menu item is of the type <code>acSubMenu</code>, this property returns the menu that is attached as the submenu, or embedded menu. The embedded menu is returned as a <code>PopupMenu</code> object.</p> <p>If the menu item is not of the type <code>acSubMenu</code>, this property returns an error.</p>
Parent	<p>You can find the menu to which a menu item belongs by using the Parent property. This property returns the menu on which the menu item resides. The parent menu is returned as a <code>PopupMenu</code> object.</p>

Enabling and disabling menu items

This example creates a new menu called “TestMenu” and inserts two menu items. The second menu item is then disabled using the Enable property and the menu is displayed on the menu bar.

```
Sub Ch6_DisableMenuItem()  
    Dim currMenuGroup As AcadMenuGroup  
    Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)  
  
    ' Create the new menu  
    Dim newMenu As AcadPopupMenu  
    Set newMenu = currMenuGroup.Menus.Add("TestMenu")  
  
    ' Add two menu items and a menu separator to the new menu  
    Dim MenuEnable As AcadPopupMenuItem  
    Dim MenuDisable As AcadPopupMenuItem  
    Dim MenuSeparator As AcadPopupMenuItem  
    Dim openMacro As String  
  
    ' Assign the macro the VB equivalent of "ESC ESC _open "  
    openMacro = Chr(vbKeyEscape) + Chr(vbKeyEscape) + "_open "  
  
    Set MenuEnable = newMenu.AddMenuItem _  
        (newMenu.count + 1, "OpenEnabled", openMacro)  
    Set MenuSeparator = newMenu.AddSeparator("")  
    Set MenuDisable = newMenu.AddMenuItem _  
        (newMenu.count + 1, "OpenDisabled", openMacro)  
  
    ' Disable the second menu item  
    MenuDisable.Enable = False  
  
    ' Display the menu on the menu bar  
    newMenu.InsertInMenuBar _  
        (ThisDrawing.Application.menuBar.count + 1)  
End Sub
```

Creating and Editing Toolbars

Using AutoCAD ActiveX/VBA you can create and edit toolbars within an existing menu group.

For more information about working with toolbars, see “Toolbars” in chapter 5, “Custom Menus,” in the *Customization Guide*.

Creating New Toolbars

To create a new toolbar, use the Add method to add a new Toolbar object to the Toolbars collection.

The Add method requires as input the name of the toolbar to add. The name is a string of alphanumeric characters with no punctuation other than a dash (-) or an underscore (_). The name is the easiest way of identifying the toolbar within the collection.

You can change the name of a toolbar once it has been created. To change the name of an existing toolbar, use the Name property for that toolbar.

Creating a new toolbar

This example creates a new toolbar called “TestToolbar” in the first menu group in the MenuGroups collection.

```
Sub Ch6_CreateTool bar()  
    Dim currMenuGroup As AcadMenuGroup  
    Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)  
  
    ' Create the new toolbar  
    Dim newTool bar As AcadTool bar  
    Set newTool bar = currMenuGroup.Tool bars.Add("TestTool bar")  
End Sub
```

Adding New Toolbar Buttons to a Toolbar

To add a new toolbar button to a toolbar use the AddToolbarButton method. This method creates a new ToolbarItem object and adds it to the designated toolbar. You should only add buttons to a toolbar while the toolbar is visible.

The AddToolbarButton method takes five parameters as input: Index, Name, HelpString, Macro, and FlyoutButton.

Index

The Index parameter is an integer that specifies the position of the new Toolbar item within the toolbar. The index begins with position zero (0) as the first position on the toolbar after the title. To add the new toolbar button to the end of a toolbar, set the Index parameter equal to the Count property of the toolbar. (The Count property of the toolbar represents the total number of toolbar buttons on that toolbar.)

Once a toolbar button has been created, you cannot change the index of the button through the Index property. To change the index of an existing toolbar button, you must delete and re-add the toolbar button to a different position, or add or delete surrounding toolbar buttons until a proper placement is achieved.

Name	<p>A name is a string that identifies the toolbar button. The string must comprise alphanumeric characters with no punctuation other than a dash (–) or an underscore (_). This string is displayed as the tooltip when the cursor is placed over the toolbar button.</p> <p>Once a toolbar button has been created, you can change the name using the Name property.</p>
Help String	<p>A help string is the text string that appears in the AutoCAD status line when a user highlights a menu item for selection.</p> <p>Once a toolbar button has been created, you can change the help string for the button using the HelpString property.</p>
Macro	<p>A macro is a series of commands that executes specific actions when a toolbar button is selected. Toolbar macros can be simply recordings of keystrokes that accomplish a task, or they can be a complex combination of commands, AutoLISP, DIESEL, or ActiveX programming code. For more information about Toolbar macros, see “Menu Macros” in chapter 5, “Custom Menus,” in the <i>Customization Guide</i>.</p> <p>Once a Toolbar button has been created, you can change the macro for the button using the Macro property.</p>
FlyoutButton	<p>The FlyoutButton parameter is an optional flag stating whether or not the new button is to be a flyout button. If the new button is to be a flyout button, this parameter must be set to TRUE. If the new button is not to be a flyout button, this parameter can be set to FALSE or it can be ignored.</p>

Adding buttons to a new toolbar

This example creates a new toolbar and adds a button to the toolbar. The button is assigned a macro that will execute the OPEN command when the button is selected.

```

Sub Ch6_AddButton()
    Dim currMenuGroup As AcadMenuGroup
    Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)

    ' Create the new toolbar
    Dim newToolBar As AcadToolBar
    Set newToolBar = currMenuGroup.Toolbars.Add("TestToolBar")

    ' Add a button to the new toolbar
    Dim newButton As AcadToolBarItem
    Dim openMacro As String

    ' Assign the macro the VB equivalent of "ESC ESC _open "
    openMacro = Chr(vbKeyEscape) + Chr(vbKeyEscape) + "_open "
    Set newButton = newToolBar.AddToolBarButton _
        ("", "NewButton", "Open a file.", openMacro)
End Sub

```

Adding Separators to a Toolbar

To add a separator to a toolbar use the `AddSeparator` method. This method creates a new `ToolBarItem` object and adds it to the designated toolbar. This kind of `ToolBarItem` object is assigned the type of `acSeparator`. The type of a `ToolBar` button can be found through the `Type` property.

The `AddSeparator` method takes one parameter as input: `Index`. The `Index` parameter is an integer that specifies the position of the separator within the toolbar. The index begins with position zero (0) as the first position on the toolbar after the title.

Defining the Toolbar Button Image

To define the images to be used on a toolbar button, use the `SetBitmaps` and `GetBitmaps` methods.

The `SetBitmaps` method takes two parameters: `SmallIconName` and `LargeIconName`.

`SmallIconName`

The small icon name identifies the ID string of the small-image resource (16 × 15 bitmap). The string must comprise alphanumeric characters with no punctuation other than a dash (–) or an underscore (_), and should include the *.bmp* extension. The resource can be either a system bitmap or a user-defined bitmap. User-defined bitmaps must be of the appropriate size and must reside in the Support path.

LargelconName

The large icon name identifies the ID string of the large-image resource (24 × 22 bitmap). The string must comprise alphanumeric characters with no punctuation other than a dash (–) or an underscore (_), and should include the *.bmp* extension. The resource can be either a system bitmap or a user-defined bitmap. User-defined bitmaps must be of the appropriate size and must reside in the Support path.

Query an existing toolbar to find the name of the icons for the buttons

```
Sub Ch6_GetButtonImages()  
    Dim Button As AcadToolBarItem  
    Dim Toolbar0 As AcadToolBar  
    Dim MenuGroup0 As AcadMenuGroup  
    Dim SmallButtonName As String  
    Dim LargeButtonName As String  
    Dim msg As String  
    Dim ButtonType As String  
  
    ' Get the first toolbar in the first menu group  
    Set MenuGroup0 = ThisDrawing.Application.MenuGroups.Item(0)  
    Set Toolbar0 = MenuGroup0.Toolbars.Item(0)  
  
    ' Clear the string variables  
    SmallButtonName = ""  
    LargeButtonName = ""  
  
    ' Create a header for the message box and  
    ' display the toolbar to be queried  
    msg = "Toolbar: " + Toolbar0.Name + vbCrLf  
    Toolbar0.Visible = True
```

```

' Iterate through the toolbar and collect data
' for each button in the toolbar. If the toolbar is
' a normal button or a flyout, collect the small
' and large button names for the button.
For Each Button In Toolbar0
    ButtonType = Choose(Button.Type + 1, "Button", _
        "Separator", "Control", "Flyout")
    msg = msg & ButtonType & ":", " "
    If Button.Type = acToolbarButton Or _
        Button.Type = acToolbarFlyout Then
        Button.GetBitmaps SmallButtonName, _
            LargeButtonName
        msg = msg + SmallButtonName + ", " _
            + LargeButtonName
    End If
    msg = msg + vbCrLf
Next Button

' Display the results
MsgBox msg
End Sub

```

Creating Flyout Toolbars

To add a flyout toolbar button to a toolbar, use the `AddToolbarButton` method. This method creates a new `ToolbarItem` object and adds it to the designated toolbar.

The `AddToolbarButton` method takes five parameters as input: `Index`, `Name`, `HelpString`, `Macro`, and `FlyoutButton`. By setting the parameter `Flyout` to `TRUE`, the new button will be created as a flyout button. The return value from this method will be the new flyout toolbar. The flyout toolbar can then be populated as a normal toolbar would be.

For more information about populating a toolbar, see “Adding New Toolbar Buttons to a Toolbar” on page 213.

Creating a flyout toolbar button

This example creates two toolbars. The first toolbar contains a flyout button. The second toolbar is attached to the flyout button on the first toolbar.

```

Sub Ch6_AddFlyoutButton()
    Dim currMenuGroup As AcadMenuGroup
    Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)

    ' Create the first toolbar
    Dim FirstToolBar As AcadToolBar
    Set FirstToolBar = currMenuGroup.ToolBars.Add("FirstToolBar")

    ' Add a flyout button to the first menu on the menu bar
    Dim FlyoutButton As AcadToolBarItem
    Set FlyoutButton = FirstToolBar.AddToolBarButton _
        ("", "Flyout", "Demonstrates a flyout button", _
        "OPEN", True)

    ' Create the second toolbar. This will be attached to
    ' the first toolbar via the flyout button.
    Dim SecondToolBar As AcadToolBar
    Set SecondToolBar = currMenuGroup.ToolBars.Add("SecondToolBar")

    ' Add a button to the next toolbar
    Dim newButton As AcadToolBarItem
    Dim openMacro As String

    ' Assign the macro the VB equivalent of "ESC ESC _open "
    openMacro = Chr(vbKeyEscape) + Chr(vbKeyEscape) + "_open "
    Set newButton = SecondToolBar.AddToolBarButton _
        ("", "NewButton", "Open a file.", openMacro)

    ' Attach the second toolbar to the flyout
    ' button on the first toolbar
    FlyoutButton.AttachToolBarToFlyout currMenuGroup.Name, _
        SecondToolBar.Name

    ' Display the first toolbar, hide the second toolbar
    FirstToolBar.Visible = True
    SecondToolBar.Visible = False
End Sub

```

Floating and Docking Toolbars

Toolbars can be docked or floated programmatically.

To float a toolbar, use the **Float** method for the toolbar. The **Float** method takes three parameters as input: **Top**, **Left**, and **NumberOfRows**. The **Top** and **Left** parameters specify the pixel location for the top and left edge of the toolbar. The **NumberOfRows** specifies the number of rows with which to create a horizontal toolbar. This number must be equal to or greater than one. The buttons of the toolbar will be distributed equally across the number of rows specified. For vertically aligned toolbars, this value specifies the number of columns.

To dock a toolbar, use the Dock method for the toolbar. The Dock method takes three parameters as input: Side, Row, and Column. The Side parameter specifies the side of the toolbar that you will be positioning in the docking maneuver. You can specify the top, bottom, left, or right side of the toolbar. The Row and Column parameters specify a number on the existing rows and columns of docked toolbars at which to dock the toolbar.

You can query a toolbar to see if it is docked by using the DockStatus property. The DockStatus property will return TRUE if the toolbar is docked and FALSE if the toolbar is floating.

Docking a toolbar

This example creates a new toolbar with three buttons on it. The toolbar is then displayed and docked on the left side of the screen.

```
Sub Ch6_DockToolbar()
    Dim currMenuGroup As AcadMenuGroup
    Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)

    ' Create the new toolbar
    Dim newToolbar As AcadToolbar
    Set newToolbar = currMenuGroup.Toolbars.Add("TestToolbar")

    ' Add three buttons to the new toolbar.
    ' All three buttons will have the same macro attached.
    Dim newButton1 As AcadToolbarItem
    Dim newButton2 As AcadToolbarItem
    Dim newButton3 As AcadToolbarItem
    Dim openMacro As String

    ' Assign the macro the VB equivalent of "ESC ESC _open "
    openMacro = Chr(vbKeyEscape) + Chr(vbKeyEscape) + "_open "

    Set newButton1 = newToolbar.AddToolbarButton(
        "", "NewButton1", "Open a file.", openMacro)
    Set newButton2 = newToolbar.AddToolbarButton(
        "", "NewButton2", "Open a file.", openMacro)
    Set newButton3 = newToolbar.AddToolbarButton(
        "", "NewButton3", "Open a file.", openMacro)

    ' Display the toolbar
    newToolbar.Visible = True

    ' Dock the toolbar to the left of the screen.
    newToolbar.Dock acToolbarDockLeft
End Sub
```

Deleting Toolbar Buttons from a Toolbar

To remove toolbar buttons from a toolbar, use the Delete method found on the toolbar button. You should only delete buttons from a toolbar while the toolbar is visible.

Exploring the Properties of Toolbar Items

All toolbar items share the following properties:

Tagstring	<p>A tag, or name tag, is a string consisting of alphanumeric and underscore (_) characters. This string uniquely identifies the toolbar item within a given toolbar. A new tag is assigned automatically when a toolbar item is created.</p> <p>You can read or write the value of a tag by using the Tagstring property.</p>
Name	<p>A name is a string identifying the toolbar item. It is also the string used for the tooltip text, which is the text string that pops up in AutoCAD when a user holds the mouse or another pointing device over the toolbar item.</p> <p>You can read or write the value of a name by using the Name property.</p>
Macro	<p>A macro is a series of commands that executes specific actions when a toolbar item is selected. Macros can simply be recordings of keystrokes that accomplish a task, or they can be a complex combination of commands, AutoLISP, DIESEL, or ActiveX programming code. For more information about macros, see “Menu Macros” in chapter 5, “Custom Menus,” in the <i>Customization Guide</i>.</p> <p>You can read or write the value of a macro by using the Macro property.</p>
HelpString	<p>A help string is the text string that appears in the AutoCAD status line for a toolbar button.</p> <p>You can read or write the value of a help string by using the HelpString property.</p>

Index

The index of a toolbar item specifies the position of that toolbar item on the toolbar to which it belongs. The index position of a toolbar always begins with position 0. For example, if the item is the first item on a toolbar, it will have an index position of 0. If it is the second item on a toolbar, it will have an index position of 1, and so on.

You can read the index position of a toolbar item by using the Index property.

Type

A toolbar item can be one of the following types: a regular toolbar button, a separator, a flyout toolbar button, or a special control element. If the item is a regular toolbar button, this property returns `acButton`. If the item is a separator, this property returns `acTool ButtonSeparator`. If the item is a flyout button, this property returns `acFlyout`. If the item is a special control element, this property returns `acControl`.

You can determine the type of a toolbar item by using the Type property.

Flyout

If the toolbar item is of the type `acFlyout`, this property returns the toolbar that is attached as the flyout toolbar. The flyout toolbar is returned as a `Toolbar` object.

If the menu item is not of the type `acFlyout`, this property returns `NULL`.

You can find the flyout toolbar of a toolbar item by using the Flyout property.

Parent

This property returns the toolbar on which the toolbar item resides. The Parent toolbar is returned as a `Toolbar` object.

You can find the toolbar to which a toolbar item belongs by using the Parent property.

Toolbar Properties

There are other properties that apply to all toolbar items on the toolbar. Such properties include: whether the toolbar is docked or floating, visible or hidden, and whether the toolbar uses large buttons or small buttons.

Creating Macros

A macro is a series of commands that executes specific actions when a toolbar item is selected. Macros can simply be recordings of keystrokes that accomplish a task, or they can be a complex combination of commands, AutoLISP, DIESEL, or ActiveX programming code.

If you intend to include command parameters in a menu macro, you must know the sequence in which that command expects its parameters. Every character in a menu macro is significant, even the blank spaces. As AutoCAD is revised and enhanced, the sequence of prompts for various commands (and sometimes even the command names) might change. Therefore, your custom menus might require minor changes when you upgrade to a new release of AutoCAD.

When command input comes from a menu item, the settings of the PICKADD and PICKAUTO system variables are assumed to be 1 and 0, respectively. This preserves compatibility with previous releases of AutoCAD and makes customization easier because you are not required to check the settings of these variables.

For more information about menu macros, see “Menu Macros” in chapter 5, “Custom Menus,” in the *Customization Guide*.

Macro Characters Mapped to ASCII Equivalents

The following table provides a synopsis of special characters used in menu macros and their equivalent ASCII numbers as they are used in VB and VBA. Use the ASCII equivalent for these special characters when creating the string for the Macro property.

Special characters used in menu and toolbar macros		
Character	ASCII equivalent	Description
;	chr(59)	Issues ENTER
^M	chr(13)	Issues ENTER
^	chr(94) + chr(124)	Issues TAB
SPACEBAR	chr(32)	Enters a space; blank space between command sequences in a menu item is equivalent to pressing the SPACEBAR

Special characters used in menu and toolbar macros *(continued)*

Character	ASCII equivalent	Description
\	chr(92)	Pauses for user input
_	chr(95)	Translates AutoCAD commands and key words that follow
+	chr(43)	Continues menu macro to the next line (if last character)
=*	chr(61) + chr(42)	Displays the current top-level image, pull-down, or shortcut menu
*^C^C	chr(42) + chr(3) + chr(3)	Prefix for a repeating item
\$	chr(36)	Loads a menu section or introduces a conditional DIESEL macro expression
^B	chr(2)	Toggles Snap on or off (CTRL+B)
^C	chr(3)	Cancels command (CTRL+C)
ESC	chr(3)	Cancels command (ESC)
^D	chr(4)	Toggles Coords on or off (CTRL+D)
^E	chr(5)	Sets the next isometric plane (CTRL+E)
^G	chr(7)	Toggles Grid on or off (CTRL+G)
^H	chr(8)	Issues backspace
^O	chr(15)	Toggles Ortho on or off (CTRL+O)
^P	chr(16)	Toggles MENU ECHO on or off
^Q	chr(17)	Echoes all prompts, status listings, and input to the printer (CTRL+Q)
^T	chr(20)	Toggles Tablet on or off (CTRL+T)
^V	chr(22)	Changes current viewport (CTRL+V)
^Z	chr(26)	Null character that suppresses the automatic addition of SPACEBAR at the end of a menu item

Macro Termination

When a macro is executed, AutoCAD places a space at the end of the macro before processing the command sequence. AutoCAD processes the following menu macro as though you had entered line SPACEBAR.

l i n e

Sometimes this is undesirable; for example, the TEXT or DIM command must be terminated by ENTER, not by a space. Also, it sometimes takes more than one space (or ENTER) to complete a command, but some text editors don't let you create a line with trailing blanks. Two special conventions get around these problems.

- When a semicolon (;) appears in a macro, AutoCAD substitutes an ENTER.
- If a line ends with a control character, a backslash (\), a plus sign (+), or a semicolon (;), AutoCAD does not add a blank after it.

Look at the following macro:

```
erase \;
```

If this item simply ended with the backslash (which indicates user input), it would fail to complete the ERASE operation, because AutoCAD doesn't add a blank after the backslash. Therefore, this macro uses a semicolon (;) to force an ENTER after the user input. Here are more examples:

```
ucs
```

```
ucs ;
```

```
text \.4 0 DRAFT Inc;;;Main St.;;;Ci ty, State;
```

Selecting the first macro enters `ucs` and SPACEBAR on the command line, and the following prompt appears:

```
Enter an option [New/Move/orthoGraphic/Prev/Restore/Save/Del/Apply/?/  
World] <World>:
```

Selecting the second macro enters `ucs`, SPACEBAR, and semicolon (;) at the command line, which accepts the default value, World. No difference between the first and second item would be evident on the screen; naturally, you wouldn't put both on the same menu.

Selecting the third macro displays a prompt for a starting point and then draws the address on three lines. In the triple-semicolon (;;:), the first semicolon ends the text string, the second causes repetition of the TEXT command, and the third calls for the default placement below the previous line.

NOTE All special characters must be input using their ASCII equivalents. For a list of ASCII equivalents, see “Macro Characters Mapped to ASCII Equivalents.”

Pausing for User Input

Sometimes it is useful to accept input from the keyboard or the pointing device in the midst of a macro by placing a backslash (\) at the point where you want input.

```
circle \1  
layer off \;
```

The first macro pauses to ask the user for the center point and then reads a radius of 1 from the macro. Note that there is no space after the backslash character (\). The next macro pauses to ask the user to enter one layer name, then turns that layer off and exits the LAYER command. The LAYER command normally prompts for another operation and exits only if you press SPACEBAR (blank) or ENTER (;).

Normally, the macro resumes after one item is entered. Therefore, it isn't possible to construct a macro that accepts a variable number of inputs (as in object selection) and then continues. However, an exception is made for the SELECT command; a backslash suspends the macro until object selection has completed. For example, consider the following macro:

```
select \change previous ;properties color red ;
```

This macro uses the SELECT command to create a selection set of one or more objects. It then issues the CHANGE command, references this selection set using the Previous option, and changes the color of all selected objects to red.

Because the backslash character (\) causes a macro to pause for user input, you cannot use a backslash for any other purpose in a macro. When specifying file directory paths, use a forward slash (/) as the path delimiter: for example, */direct/file*.

The following circumstances delay resumption of a macro:

- If input of a point is expected, Object Snap modes may precede entry of the actual point.
- If X/Y/Z point filters are used, the macro remains suspended until the entire point has been accumulated.
- For the SELECT command only, the macro doesn't resume until object selection has been completed.

- If the user responds with a transparent command, the suspended macro remains suspended until the transparent command is completed and the originally requested input is received.
- If the user responds by choosing another macro (to supply options or to execute a transparent command), the original macro is suspended, and the newly selected item is processed to completion before the suspended macro is resumed.

Canceling a Command

To make sure you have no previous incomplete commands, use `^C^C` in a macro. This is the same as pressing ESC twice from the keyboard. Although a single `^C` cancels most commands, `^C^C` is required to return to the Command prompt from a DIM command. Therefore, `^C^C` ensures that AutoCAD returns to the Command prompt in most cases.

Macro Repetition

Once you have selected a command, you are likely to use it several times before moving on to another command. That is how most people use tools; you pick up a tool, do several things with it, then pick up another tool, and so on. To avoid picking up the tool before each use, AutoCAD provides a command repetition capability, triggered by a null response. However, you cannot use this feature to specify command options.

This feature makes it possible for you to repeat frequently used commands until you choose another command. If a macro begins with `*^C^C` immediately following the item label, the macro is saved in memory. Subsequent Command prompts are answered by that macro until it is terminated by ESC or by the selection of another macro.

Do not use `^C` (Cancel) within a macro that begins with the string `*^C^C`; this cancels the macro repetition.

The following is an example of the repetitive, or modal, approach to command handling.

```
*^C^CMOVE Single  
*^C^CCOPY Single  
*^C^CERASE Single  
*^C^CSTRETCH Single Crossing  
*^C^CROTATE Single  
*^C^CSCALE Single
```

Macro repetition does not work for items in image tile menus.

Use of Single Object Selection Mode

Single object selection puts object selection in single selection mode, disables the normal dialog conducted by object selection, and causes the selection to return the first object(s) selected by a subsequent option. This can be quite handy in a macro. For example, consider the following macro:

```
*^C^CERASE single
```

This macro terminates the current command and activates the ERASE command with the single selection option. After you select this item, you either point to the single object to be erased or point to a blank area and specify a window. The object(s) selected in this way are erased, and the macro is repeated (due to the leading asterisk) so that you can erase something else. Single selection mode leads to more dynamic interaction with AutoCAD.

Creating Status-Line Help for Menu Items and Toolbar Items

Status-line help messages are an important aspect of native help support. These are the simple, descriptive messages that appear in the status line when a menu or toolbar item is highlighted. The status-line help for all menu and toolbar items is contained in the `HelpString` property for that item.

The `HelpString` property is empty when the menu or toolbar item is first created.

Adding status-line help to a menu item

This example creates a new menu called “TestMenu” and then creates a menu item called “Open.” The menu item is then assigned status-line help via the HelpString property.

```
Sub Ch6_AddHelp()  
    Dim currMenuGroup As AcadMenuGroup  
    Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)  
  
    ' Create the new menu  
    Dim newMenu As AcadPopupMenu  
    Set newMenu = currMenuGroup.Menus.Add _  
        ("Te" + Chr(Asc("&"))) + "stMenu")  
  
    ' Add a menu item to the new menu  
    Dim newMenuItem As AcadPopupMenu.Item  
    Dim openMacro As String  
    ' Assign the macro the VBA equivalent of "ESC ESC _open "  
    openMacro = Chr(vbKeyEscape) + Chr(vbKeyEscape) + "_open "  
  
    ' Create the menu item  
    Set newMenuItem = newMenu.AddMenuItem _  
        (newMenu.count + 1, Chr(Asc("&"))) _  
        + "Open", openMacro)  
  
    ' Add the status line help to the menu item  
    newMenuItem.HelpString = "Opens an AutoCAD drawing file."  
  
    ' Display the menu on the menu bar  
    newMenu.InsertInMenuBar _  
        (ThisDrawing.Application.menuBar.count + 1)  
End Sub
```

Adding Entries to the Right-Click Menu

The right-click menu, or shortcut menu, is a special menu included in the AutoCAD base menu group. This menu appears when the user holds SHIFT and clicks the right mouse button.

AutoCAD finds the shortcut menu by looking in the base menu group for a menu with the ShortcutMenu property equal to TRUE. You can add new menu items to the shortcut menu by following the steps listed in “Adding New Menu Items to a Menu” on page 204.

New menu groups may or may not have a shortcut menu available. To create a shortcut menu for a menu group, follow the guidelines listed in “Creating New Menus” on page 203, and use POP0 as the label for the new menu.

Adding a menu item to the end of the right-click menu

This example adds the menu item “OpenDWG” to the end of the right-click menu.

```
Sub Ch6_AddMenuItemToShortcutMenu()  
    Dim currMenuGroup As AcadMenuGroup  
    Set currMenuGroup = ThisDrawing.Application.MenuGroups.Item(0)  
  
    ' Find the shortcut menu and assign it to the  
    ' shortcutMenu variable  
    Dim scMenu As AcadPopupMenu  
    Dim entry As AcadPopupMenu  
    For Each entry In currMenuGroup.Menus  
        If entry.ShortcutMenu = True Then  
            Set scMenu = entry  
        End If  
    Next entry  
  
    ' Add a menu item to the shortcut menu  
    Dim newItem As AcadPopupMenu  
    Dim openMacro As String  
    ' Assign the macro the VBA equivalent of "ESC ESC _open "  
    openMacro = Chr(vbKeyEscape) + Chr(vbKeyEscape) + "_open "  
  
    Set newItem = scMenu.AddMenuItem _  
        ("", Chr(Asc("&"))) _  
        + "OpenDWG", openMacro)  
End Sub
```


Using Events

7

Events are notifications, or messages, that are sent out by AutoCAD to inform you about the current state of the session, or alert you that something has happened. For example, when a drawing is opened the `BeginOpen` event is triggered. This event contains the name of the AutoCAD drawing that was opened. There is another event triggered when a drawing is closed. Given this information you could write a subroutine, or event handler, that uses these events to track the amount of time a user spends working on a particular drawing.

In this chapter

- Understanding the Events in AutoCAD
- Guidelines for Writing Event Handlers
- Handling Application Level Events
- Handling Document Level Events
- Handling Object Level Events

Understanding the Events in AutoCAD

There are three types of events in AutoCAD:

- Application level events respond to changes in the AutoCAD application and its environment. These events respond to the opening, saving, closing and printing of drawings, creation of new drawings, issuing of AutoCAD commands, loading or unloading of ARX and LISP applications, changes to system variables, and changes to the Application window.
- Document level events respond to the changes of a specific document or its contents. These events respond to the addition, deletion, or modification of objects, activation of a shortcut menu, changes in the pickfirst selection set, changes to the Drawing window, and regeneration of the drawing. There are also document level events for the opening, closing, and printing of a drawing, and the loading or unloading of ARX and LISP applications from the drawing.
- Object level events respond to the changes of a specific object. Currently there is only one object level event. It is triggered whenever an object is modified.

Subroutines that respond to events are called event handlers and are executed automatically each and every time their designated event is triggered. Information contained in events, such as the drawing name in the BeginOpen event, are passed to event handlers through parameters.

Guidelines for Writing Event Handlers

It is important to remember that events simply provide information on the state or activities taking place within AutoCAD. Although event handlers can be written to respond to those events, AutoCAD is often in the middle of processing commands when the event handler is triggered. Event handlers, therefore, have some restrictions on what they can do if they are to provide safe operations in conjunction with AutoCAD and its database.

- Do not rely on the sequence of events.

When writing event handlers, do not rely on the sequence of events to happen in the exact order you think they occur. For example, if you issue an OPEN command, the events BeginCommand, BeginOpen, EndOpen, and EndCommand will all be triggered. However, they may not occur in that order. The only event sequence you can safely rely on is that a Begin event will occur before the corresponding End event. In the previous

example, the events may get triggered in the following order: BeginCommand, BeginOpen, EndCommand, and EndOpen, or even BeginCommand, EndCommand, BeginOpen, and EndOpen.

- Do not rely on the sequence of operations.

If you delete object1 and then object2, do not rely on the fact that you will receive the ObjectErased event for object1 and then for object2. You may receive the ObjectErased event for object2 first.

- Do not attempt any interactive functions from an event handler.

Attempting to execute interactive functions from within an event handler can cause serious problems, as AutoCAD may still be processing a command at the time the event is triggered. Therefore, you should always avoid the use of input-acquisition methods such as GetPoint, GetEntity, GetKeyword, and so on, as well as selection set operations and the SendCommand method from within event handlers.

- Do not launch a dialog box from within an event handler.

Dialog boxes are considered interactive functions and can interfere with the current operation of AutoCAD. Message boxes and alert boxes are not considered interactive and can be issued safely, however issuing a message box within an event handler for the BeginModal, EndModal, Activate, Deactivate, and BeginRightClick events results in unexpected sequencing.

- You can write data to any object in the database, except the object that issued the event.

Obviously any object causing an event to be triggered could still be open for use with AutoCAD and the operation currently in progress. Therefore, avoid writing any information to an object from an event handler for the same object. However, you can safely read information from the object triggering an event. For example, suppose you have a floor that is filled with tiles and you create an event handler attached to the border of the floor. If you change the size of the floor, the event handler will automatically add or subtract tiles to fill the new area. The event handler will be able to read the new area of the border, but it cannot attempt any changes on the border itself.

- Do not perform any action from an event handler that will trigger the same event.

If you perform the same action in an event handler that triggers that same event, you will create an infinite loop. For example, you should never attempt to open a drawing from within the BeginOpen event, or AutoCAD will simply continue to open more drawings until the maximum number of open drawings is reached.

- Remember that no events will be fired while AutoCAD is displaying a modal dialog.

Handling Application Level Events

Application level events are not persistent in AutoCAD VBA. That is, they are not automatically enabled when a VBA project is loaded. Application level events must therefore be enabled for VBA and all other ActiveX Automation controllers.

Once the application level events are enabled, you have a wide range of events available to you. These events include:

<code>AppActivate</code>	Triggered just before the main Application window is activated.
<code>AppDeactivate</code>	Triggered just before the main Application window is deactivated.
<code>ARXLoaded</code>	Triggered when an ObjectARX application has been loaded.
<code>ARXUnloaded</code>	Triggered when an ObjectARX application has been unloaded.
<code>BeginCommand</code>	Triggered immediately after a command is issued, but before it completes.
<code>BeginFileDrop</code>	Triggered when a file is dropped on the main Application window.
<code>BeginLISP</code>	Triggered immediately after AutoCAD receives a request to evaluate a LISP expression.
<code>BeginModal</code>	Triggered just before a modal dialog box is displayed.
<code>BeginOpen</code>	Triggered immediately after AutoCAD receives a request to open an existing drawing.
<code>BeginPlot</code>	Triggered immediately after AutoCAD receives a request to print a drawing.
<code>BeginQuit</code>	Triggered just before an AutoCAD session ends.
<code>BeginSave</code>	Triggered immediately after AutoCAD receives a request to save the drawing.
<code>EndCommand</code>	Triggered immediately after a command completes.
<code>EndLISP</code>	Triggered upon completion of evaluating a LISP expression.

EndModal	Triggered just after a modal dialog box is dismissed.
EndOpen	Triggered immediately after AutoCAD finishes opening an existing drawing.
EndPlot	Triggered after a document has been sent to the printer.
EndSave	Triggered when AutoCAD has finished saving the drawing.
LISPCancelled	Triggered when the evaluation of a LISP expression is canceled.
NewDrawing	Triggered just before a new drawing is created.
SysVarChanged	Triggered when the value of a system variable is changed.
WindowChanged	Triggered when there is a change to the Application window.
WindowMovedOrResized	Triggered just after the Application window has been moved or resized.

Enabling Application Level Events

Before you can use application level events you must create a new class module and declare an object of type AcadApplication with events. For example, assume that a new class module is created and called EventClassModule. The new class module contains the declaration of the application with the VBA keyword WithEvents.

To create a new class and declare an Application object with events:

- 1 In the VBA IDE, insert a class module. From the Insert menu, choose Class Module.
- 2 Select the new class module in the Project window.
- 3 Change the name of the class in the Properties window to EventClassModule.
- 4 Open the Code window for the class using F7, or by selecting the menu option View ► Code.
- 5 In the Code window for the class, add the following line:

```
Public WithEvents App As AcadApplication
```

After the new object has been declared with events, it appears in the Object drop-down list box in the class module, and you can write event procedures for the new object in the class module. (When you select the new object in the Object box, the valid events for that object are listed in the Procedure drop-down list box.)

Before the procedures will run, however, you must connect the declared object in the class module with the Application object. You can do this with the following code from any module.

To connect the declared object to the Application object:

- 1 In the Code window for your main module, add the following line to the declarations section:

```
Dim X As New EventClassModule
```

- 2 In the same window, add the following subroutine:

```
Sub InitializeEvents()  
    Set X.App = ThisDrawing.Application  
End Sub
```

- 3 In the code for your main module, add a call to the InitializeEvents subroutine:

```
Call InitializeEvents
```

Once the InitializeEvents procedure has been run, the App object in the class module points to the Application object specified, and any event procedures in the class module will run when the events occur.

Prompting to continue when a drawing is dropped into AutoCAD

This example intercepts the load process when a file has been dragged and dropped into AutoCAD. A message box containing the name of the file that was dropped and Yes/No/Continue buttons that allow the user to decide if the file should continue to be loaded display. If the user chooses to cancel out of the operation, that decision is returned through the Cancel parameter of the BeginFileDrop event and the file is not loaded.

```
Public WithEvents ACADApp As AcadApplication
```

```
Sub Example_AcadApplication_Events()  
    ' This example initializes the public variable (ACADApp)  
    ' which will be used to intercept AcadApplication Events  
    ' Run this procedure FIRST!
```



```

' We could get the application from the ThisDocument
' object, but that would require having a drawing open,
' so we grab it from the system.
Set ACADApp = GetObject(, "AutoCAD.Application")
End Sub

Private Sub ACADApp_BeginFileDrop _
    (ByVal FileName As String, Cancel As Boolean)
' This example intercepts an Application BeginFileDrop event.
'
' This event is triggered when a drawing file is dragged
' into AutoCAD.
'
' To trigger this example event:
' 1) Make sure to run the example that initializes
' the public variable (named ACADApp) linked to this event.
'
' 2) Drag an AutoCAD drawing file into the AutoCAD
' application from either the Windows Desktop
' or Windows Explorer
'
' Use the "Cancel" variable to stop the loading of the
' dragged file, and the "FileName" variable to notify
' the user which file is about to be dragged in.

If MsgBox("AutoCAD is about to load " & FileName & vbCrLf _
    & "Do you want to continue loading this file?", _
    vbYesNoCancel + vbQuestion) <> vbYes Then
    Cancel = True
End If
End Sub

```

Handling Document Level Events

Document level events are persistent in AutoCAD VBA. That is, they are automatically enabled when a VBA project is loaded. However, they are not enabled for any other controller, such as VB. Document level events must therefore be enabled for all other ActiveX Automation controllers.

Once the document level events are enabled, you have a wide range of events available to you. These events include

Activate	Triggered when a Document window is activated.
BeginClose	Triggered just before a document is closed.
BeginCommand	Triggered immediately after a command is issued, but before it completes.

BeginDoubleClick

Triggered after the user double-clicks on an object in the drawing.

BeginLISP

Triggered immediately after AutoCAD receives a request to evaluate a LISP expression.

BeginPlot

Triggered immediately after AutoCAD receives a request to print a drawing.

BeginRightClick

Triggered after the user right-clicks on the Drawing window.

BeginSave

Triggered immediately after AutoCAD receives a request to save the drawing.

BeginShortcutMenuCommand

Triggered after the user right-clicks on the Drawing window, and before the shortcut menu appears in Command mode.

BeginShortcutMenuDefault

Triggered after the user right-clicks on the Drawing window, and before the shortcut menu appears in Default mode.

BeginShortcutMenuEdit

Triggered after the user right-clicks on the Drawing window, and before the shortcut menu appears in Edit mode.

BeginShortcutMenuGrip

Triggered after the user right-clicks on the Drawing window, and before the shortcut menu appears in Grip mode.

BeginShortcutMenuOsnap

Triggered after the user right-clicks on the Drawing window, and before the shortcut menu appears in Osnap mode.

Deactivate	Triggered when the Drawing window is deactivated.
EndCommand	Triggered immediately after a command completes.
EndLISP	Triggered upon completion of evaluating a LISP expression.
EndPlot	Triggered after a document has been sent to the printer.
EndSave	Triggered when AutoCAD has finished saving the drawing.
EndShortcutMenu	Triggered after the shortcut menu appears.
LayoutSwitched	Triggered after the user switches to a different layout.
LISPCancelled	Triggered when the evaluation of a LISP expression is canceled.
ObjectAdded	Triggered when an object has been added to the drawing.
ObjectErased	Triggered when an object has been erased from the drawing.
ObjectModified	Triggered when an object in the drawing has been modified.
SelectionChanged	Triggered when the current pickfirst selection set changes.
WindowChanged	Triggered when there is a change to the Document window.
WindowMovedOrResized	Triggered just after the Drawing window has been moved or resized.

Enabling Document Level Events in Environments Other Than VBA

Before you can use document level events in VB or another environment besides VBA, you must create a new class module and declare an object of type AcadDocument with events. For example, assume a new class module is created and called EventClassModule. The new class module contains the declaration of the application with the VBA keyword WithEvents.

To create a new class and declare a Document object with events:

- 1 In the VBA IDE, insert a class module. From the Insert menu, choose Class Module.
- 2 Select the new class module in the Project window.
- 3 Change the name of the class in the Properties window to EventClassModule.
- 4 Open the Code window for the class using F7, or by selecting the menu option View ► Code.
- 5 In the Code window for the class, add the following line:

```
Public WithEvents Doc As AcadDocument
```

After the new object has been declared with events, it appears in the Object drop-down list box in the class module, and you can write event procedures for the new object in the class module. (When you select the new object in the Object box, the valid events for that object are listed in the Procedure drop-down list box.)

Before the procedures will run, however, you must connect the declared object in the class module with the Document object. You can do this with the following code from any module.

To connect the declared object to the Document object:

- 1 In the Code window for your main module, add the following line to the declarations section:

```
Dim X As New EventClassModule
```

- 2 In the same window, add the following subroutine:

```
Sub InitializeEvents()  
    Set X.Doc = ThisDrawing  
End Sub
```

3 In the code for your main module, add a call to the `InitializeApp` subroutine:

```
Call InitializeEvents
```

Once the `InitializeEvents` procedure has been run, the `Doc` object in the class module points to the Document object created, and any event procedures in the class module will run when the events occur.

Coding Document Level Events in Environments Other Than VBA

Once the document level events have been enabled, you will find the `Doc` class variable available from the Object drop-down list of the Class Module Code window. Select the `Doc` class and the list of available events will appear in the Procedure drop-down list. Simply select the event you want to write a handler for and the handler skeleton is created automatically.

Coding Document Level Events in VBA

As mentioned in “Handling Document Events,” document level events are automatically enabled when a VBA project is loaded. To write event handlers for document level events in VBA, you simply select `AcadDocument` from the Object drop-down list in the Code window. The available events for the document will appear in the Procedure drop-down list. Simply select the event you want to write a handler for and the handler skeleton is created automatically.

Note that event handlers created in this fashion apply to the current active drawing. To create event handlers for a specific drawing, first follow the steps listed in “Enabling Document Level Events in Environments Other Than VBA” on page 240. This will allow you to enable a specific document for events.

Updating the shortcut menu at the `BeginShortcutMenuDefault` and `EndShortcutMenu` events

This example uses the event handler for the `BeginShortcutMenuDefault` event to add the “OpenDWG” menu item to the beginning of the shortcut menu. Then the event handler for the `EndShortcutMenu` event removes the additional menu item so that it is not saved permanently in the user’s menu configuration.

```

Private Sub AcadDocument_BeginShortcutMenuDefault _
    (ShortcutMenu As AutoCAD.IAcadPopupMenu)
    On Error Resume Next
    ' Add a menu item to the cursor menu
    Dim newItem As AcadPopupMenu
    Dim openMacro As String
    openMacro = Chr(vbKeyEscape) + Chr(vbKeyEscape) + "_open "
    Set newItem = ShortcutMenu.AddMenuItem _
        (0, Chr(Asc("&"))) _
        + "OpenDWG", openMacro)

End Sub

Private Sub AcadDocument_EndShortcutMenu _
    (ShortcutMenu As AutoCAD.IAcadPopupMenu)
    On Error Resume Next
    ShortcutMenu.Item("OpenDWG").Delete
End Sub

```

Handling Object Level Events

The object level events is not persistent in AutoCAD VBA. That is, it is not automatically enabled when a VBA project is loaded. An object level event must be enabled for VBA and all other ActiveX Automation controllers.

Once the object level events is enabled, the Modified event is available to you. This event is triggered when an object in the drawing has been modified.

Enabling the Object Level Event

Before you can use object level events you must create a new class module and declare an object of type AcadObject with events. For example, assume that a new class module is created and called EventClassModule. The new class module contains the declaration of the application with the VBA keyword WithEvents.

To create a new class and declare a Circle object with events:

- 1 In the VBA IDE, insert a class module. From the Insert menu, choose Class Module.
- 2 Select the new class module in the Project window.
- 3 Change the name of the class in the Properties window to EventClassModule.

4 Open the Code window for the class using F7, or by selecting the menu option View ► Code.

5 In the Code window for the class, add the following line:

```
Public WithEvents Object As AcadCircle
```

After the new object has been declared with events, it appears in the Object drop-down list box in the class module, and you can write event procedures for the new object in the class module. (When you select the new object in the Object box, the valid events for that object are listed in the Procedure drop-down list box.)

Before the procedures will run, however, you must connect the declared object in the class module with the Circle object. You can do this with the following code from any module.

To connect the declared object to the Automation object:

1 In the Code window for your main module, add the following line to the declarations section:

```
Dim X As New EventClassModule
```

2 In the same window, create a circle called “MyCircle” and initialize it as containing events:

```
Sub InitializeEvents()  
    Dim MyCircle As AcadCircle  
    Dim centerPoint(0 To 2) As Double  
    Dim radius As Double  
    centerPoint(0) = 0#: centerPoint(1) = 0#: centerPoint(2) = 0#  
    radius = 5#  
    Set MyCircle = ThisDrawing.ModelSpace.AddCircle(centerPoint,  
radius)  
    Set X.Object = MyCircle  
End Sub
```

3 In the code for your main module, add a call to the InitializeApp subroutine:

```
Call InitializeEvents
```

Once the InitializeEvents procedure has been run, the Circle object in the class module points to the Circle object created, and any event procedures in the class module will run when the events occur.

NOTE When coding in VBA, you must provide an event handler for all objects enabled for the Modified event. If you do not provide a handler, VBA may terminate unexpectedly.

Displaying the area of a closed polyline whenever the polyline is updated

This example creates a lightweight polyline with events. The event handler for the polyline then displays the new area whenever the polyline is changed. To trigger the event, simply change the size of the polyline in AutoCAD. Remember you must run the CreatePLineWithEvents subroutine before the event handler is activated.

```
Public WithEvents PLine As AcadLWPolyline

Sub CreatePLineWithEvents()
    ' This example creates a lightweight polyline
    Dim points(0 To 9) As Double
    points(0) = 1: points(1) = 1
    points(2) = 1: points(3) = 2
    points(4) = 2: points(5) = 2
    points(6) = 3: points(7) = 3
    points(8) = 3: points(9) = 2
    Set PLine = ThisDrawing.ModelSpace.AddLightweightPolyline(points)
    PLine.Closed = True
    ThisDrawing.Application.ZoomAll
End Sub

Private Sub PLine_Modified _
    (ByVal pObject As AutoCAD.IAcadObject)
    ' This event is triggered when the polyline is resized.
    ' If the polyline is deleted the modified event is still
    ' triggered, so we use the error handler to avoid
    ' reading data from a deleted object.
    On Error GoTo ERRORHANDLER
    MsgBox "The area of " & pObject.ObjectName & " is: " _
        & pObject.Area
    Exit Sub

ERRORHANDLER:
    MsgBox Err.Description
End Sub
```


Working in Three-Dimensional Space

Most drawings consist of two-dimensional (2D) views of objects that are three-dimensional (3D). Though this method of drafting is widely used in the architectural and engineering communities, it is limited: the drawings are 2D representations of 3D objects and must be visually interpreted. Moreover, because the views are created independently, there are more possibilities for error and ambiguity. As a result, you may want to create true 3D models instead of 2D representations. You can use the AutoCAD drawing tools to create detailed, realistic 3D objects and manipulate them in various ways.

8

In this chapter

- Specifying 3D Coordinates
- Defining a User Coordinate System
- Converting Coordinates
- Creating 3D Objects
- Editing in 3D
- Editing 3D Solids

Specifying 3D Coordinates

Entering 3D WCS coordinates is similar to entering 2D WCS coordinates. In addition to specifying *X* and *Y* values, you specify a *Z* value. As with the 2D coordinates, a variant is used to pass the coordinates to ActiveX methods and properties, and to query the coordinates.

For more information about specifying 3D coordinates, see “Enter 3D Coordinates” in chapter 15, “Use Precision Tools,” in the *User’s Guide*.

Defining and querying the coordinates for 2D and 3D polylines

This example creates two polylines, each with three coordinates. The first polyline is a 2D polyline, the second polyline is 3D. Notice the length of the array containing the vertices is expanded to include the *Z* coordinates in the creation of the 3D polyline. The example concludes by querying the coordinates of the polylines and displaying the coordinates in a message box.

```
Sub Ch8_Polyline_2D_3D()  
    Dim pline2DObj As AcadLWPolyline  
    Dim pline3DObj As AcadPolyline  
  
    Dim points2D(0 To 5) As Double  
    Dim points3D(0 To 8) As Double  
  
    ' Define three 2D polyline points  
    points2D(0) = 1: points2D(1) = 1  
    points2D(2) = 1: points2D(3) = 2  
    points2D(4) = 2: points2D(5) = 2  
  
    ' Define three 3D polyline points  
    points3D(0) = 1: points3D(1) = 1: points3D(2) = 0  
    points3D(3) = 2: points3D(4) = 1: points3D(5) = 0  
    points3D(6) = 2: points3D(7) = 2: points3D(8) = 0  
  
    ' Create the 2D light weight Polyline  
    Set pline2DObj = ThisDrawing.ModelSpace.  
        AddLightWeightPolyline(points2D)  
    pline2DObj.Color = acRed  
    pline2DObj.Update  
  
    ' Create the 3D polyline  
    Set pline3DObj = ThisDrawing.ModelSpace.  
        AddPolyline(points3D)  
    pline3DObj.Color = acBlue  
    pline3DObj.Update
```

```

' Query the coordinates of the polylines
Dim get2Dpts As Variant
Dim get3Dpts As Variant

get2Dpts = pline2DObj.Coordinates
get3Dpts = pline3DObj.Coordinates

' Display the coordinates

MsgBox ("2D polyline (red): " & vbCrLf & _
    get2Dpts(0) & ", " & get2Dpts(1) & vbCrLf & _
    get2Dpts(2) & ", " & get2Dpts(3) & vbCrLf & _
    get2Dpts(4) & ", " & get2Dpts(5))

MsgBox ("3D polyline (blue): " & vbCrLf & _
    get3Dpts(0) & ", " & get3Dpts(1) & ", " & _
    get3Dpts(2) & vbCrLf & _
    get3Dpts(3) & ", " & get3Dpts(4) & ", " & _
    get3Dpts(5) & vbCrLf & _
    get3Dpts(6) & ", " & get3Dpts(7) & ", " & _
    get3Dpts(8))

End Sub

```

Defining a User Coordinate System

You define a user coordinate system (UCS) object to change the location of the (0, 0, 0) origin point and the orientation of the XY plane and Z axis. You can locate and orient a UCS anywhere in 3D space, and you can define, save, and recall as many user coordinate systems as you require. Coordinate input and display are relative to the current UCS.

To indicate the origin and orientation of the UCS, you can display the UCS icon at the UCS origin point using the `UCSIconAtOrigin` property. If the UCS icon is turned on (see the `UCSIconOn` property) and is not displayed at the origin, it is displayed at the WCS coordinate defined by the `UCSORG` system variable.

You can create a new user coordinate system using the `Add` method. This method requires four values as input: the coordinate of the origin, a coordinate on the X and Y axes, and the name of the UCS.

All coordinates in the AutoCAD ActiveX Automation are entered in the world coordinate system (WCS). Use the `GetUCSMatrix` method to return the transformation matrix of a given UCS. Use this transformation matrix to find the equivalent WCS coordinates.

To make a UCS active, use the ActiveUCS property on the Document object. If changes are made to the active UCS, the new UCS object must be reset as the active UCS for the changes to appear. To reset the active UCS, simply call the ActiveUCS property again with the updated UCS object.

For more information about defining a UCS, see “Control the User Coordinate System in 3D” in chapter 15, “Use Precision Tools” in the *User’s Guide*.

Creating a new UCS, making it active, and translating the coordinates of a point into the UCS coordinates

The following subroutine creates a new UCS and sets it as the active UCS for the drawing. It then asks the user to pick a point in the drawing, and returns both WCS and UCS coordinates for the point.

```
Sub Ch8_NewUCS()
    ' Define the variables we will need
    Dim ucsObj As AcadUCS
    Dim origin(0 To 2) As Double
    Dim xAxisPnt(0 To 2) As Double
    Dim yAxisPnt(0 To 2) As Double
    ' Define the UCS points
    origin(0) = 4: origin(1) = 5: origin(2) = 3
    xAxisPnt(0) = 5: xAxisPnt(1) = 5: xAxisPnt(2) = 3
    yAxisPnt(0) = 4: yAxisPnt(1) = 6: yAxisPnt(2) = 3

    ' Add the UCS to the
    ' UserCoordinatesSystems collection
    Set ucsObj = ThisDrawing.UserCoordinateSystems. _
        Add(origin, xAxisPnt, yAxisPnt, "New_UCS")
    ' Display the UCS icon
    ThisDrawing.ActiveViewport.UCIconAtOrigin = True
    ThisDrawing.ActiveViewport.UCIconOn = True

    ' Make the new UCS the active UCS
    ThisDrawing.ActiveUCS = ucsObj
    MsgBox "The current UCS is : " & ThisDrawing.ActiveUCS.Name & _
        & vbCrLf & " Pick a point in the drawing."

    ' Find the WCS and UCS coordinate of a point
    Dim WCSPnt As Variant
    Dim UCSPnt As Variant

    WCSPnt = ThisDrawing.Utility.GetPoint(, "Enter a point: ")
    UCSPnt = ThisDrawing.Utility.TranslateCoordinates _
        (WCSPnt, acWorld, acUCS, False)

    MsgBox "The WCS coordinates are: " & WCSPnt(0) & ", " & _
        & WCSPnt(1) & ", " & WCSPnt(2) & vbCrLf & _
        "The UCS coordinates are: " & UCSPnt(0) & ", " & _
        & UCSPnt(1) & ", " & UCSPnt(2)
End Sub
```

Converting Coordinates

The `TranslateCoordinates` method translates a point or a displacement from one coordinate system to another. A point argument, called `Original Point`, can be interpreted as either a 3D point or a 3D displacement vector. This argument is distinguished by the `Boolean` argument, `Displacement`. If the `Displacement` argument is set to `TRUE` the `Original Point` argument is treated as a displacement vector; otherwise, it is treated as a point. Two more arguments determine which coordinate system the `Original Point` is from, and to which coordinate system the `Original Point` is to be converted. The following AutoCAD coordinate systems can be specified in the `From` and `To` arguments:

- WCS** World Coordinate System—the reference coordinate system. All other coordinate systems are defined relative to the WCS, which never changes. Values measured relative to the WCS are stable across changes to other coordinate systems. All points passed in and out of ActiveX methods and properties are expressed in the WCS unless otherwise specified.
- UCS** User Coordinate System—the working coordinate system. The user specifies a UCS to make drawing tasks easier. All points passed to AutoCAD commands, including those returned from AutoLISP routines and external functions, are points in the current UCS (unless the user precedes them with an * at the Command prompt). If you want your application to send coordinates in the WCS, OCS, or DCS to AutoCAD commands, you must first convert them to the UCS by calling the `TranslateCoordinates` method.
- OCS** Object Coordinate System—point values specified by certain methods and properties for the Polyline and LightweightPolyline objects are expressed in this coordinate system, relative to the object. These points are usually converted into the WCS, current UCS, or current DCS, according to the intended use of the object. Conversely, points in WCS, UCS, or DCS must be translated into an OCS before they are written to the database by means of the same properties. See the *AutoCAD ActiveX and VBA Reference* for the methods and properties that use this coordinate system.

When converting coordinates to or from the OCS you must enter the normal for the OCS in the final argument of the `TranslateCoordinates` function.

DCS

Display Coordinate System—the coordinate system where objects are transformed before they are displayed. The origin of the DCS is the point stored in the AutoCAD system variable `TARGET`, and its *Z* axis is the viewing direction. In other words, a viewport is always a plan view of its DCS. These coordinates can be used to determine where something will be displayed to the AutoCAD user.

PSDCS

Paper Space DCS—this coordinate system can be transformed only to or from the DCS of the currently active model space viewport. This is essentially a 2D transformation, where the *X* and *Y* coordinates are always scaled and offset if the `Display` argument is `FALSE`. The *Z* coordinate is scaled but never translated. Therefore, it can be used to find the scale factor between the two coordinate systems. The PSDCS can be transformed only into the current model space viewport. If the `from` argument equals PSDCS, then the `to` argument must equal DCS, and vice versa.

Translating OCS coordinates to WCS coordinates

This example creates a polyline in model space. The first vertex for the polyline is then displayed in both the OCS and WCS coordinates. The conversion from OCS to WCS requires the normal for the OCS be placed in the last argument of the `TranslateCoordinates` method.

```
Sub Ch8_TranslateCoordinates()  
    ' Create a polyline in model space.  
    Dim plineObj As AcadPolyline  
    Dim points(0 To 14) As Double  
  
    ' Define the 2D polyline points  
    points(0) = 1: points(1) = 1: points(2) = 0  
    points(3) = 1: points(4) = 2: points(5) = 0  
    points(6) = 2: points(7) = 2: points(8) = 0  
    points(9) = 3: points(10) = 2: points(11) = 0  
    points(12) = 4: points(13) = 4: points(14) = 0  
  
    ' Create a light weight Polyline object in model space  
    Set plineObj = ThisDrawing.ModelSpace.AddPolyline(points)
```

```

' Find the X and Y coordinates of the
' first vertex of the polyline
Dim firstVertex As Variant
firstVertex = plineObj.Coordinate(0)

' Find the Z coordinate for the polyline
' using the elevation property
firstVertex(2) = plineObj.Elevation

' Change the normal for the pline so that the
' difference between the coordinate systems
' is obvious.
Dim plineNormal(0 To 2) As Double
plineNormal(0) = 0#
plineNormal(1) = 1#
plineNormal(2) = 2#
plineObj.Normal = plineNormal

' Translate the OCS coordinate into WCS
Dim coordinateWCS As Variant
coordinateWCS = ThisDrawing.Utility.TranslateCoordinates _
(firstVertex, acOCS, acWorld, False, plineNormal)

' Display the coordinates of the point
MsgBox "The first vertex has the following coordinates: " _
& vbCrLf & "OCS: " & firstVertex(0) & ", " & _
firstVertex(1) & ", " & firstVertex(2) & vbCrLf & _
"WCS: " & coordinateWCS(0) & ", " & _
coordinateWCS(1) & ", " & coordinateWCS(2)
End Sub

```

Creating 3D Objects

AutoCAD supports three types of 3D modeling: wireframe, surface, and solid. Each type has its own creation and editing techniques.

For more information about creating 3D objects, see “Create 3D Objects” in chapter 16, “Draw Geometric Objects,” in the *User’s Guide*.

Creating Wireframes

With AutoCAD you can create wireframe models by positioning any 2D planar object anywhere in 3D space. You can position 2D objects in 3D space using several methods:

- Create the object by entering 3D points. You enter a coordinate that defines the X, Y, and Z location of the point.

- Set the default construction plane (XY plane) on which you will draw the object by defining a UCS.
- Move the object to its proper orientation in 3D space after you create it.

Also, you can create some wireframe objects, such as polylines, that can exist in all three dimensions. Use the `Add3DPoly` method to create 3D polylines.

For more information on creating wireframes, see “Create Wireframe Models” in the *User’s Guide*.

Creating Meshes

A rectangular mesh (`PolygonMesh` object) represents an object’s surface using planar facets. The mesh density, or number of facets, is defined in terms of a matrix of M and N vertices, similar to a grid consisting of columns and rows. M and N specify the column and row position, respectively, of any given vertex. You can create meshes in both 2D and 3D, but they are used primarily for 3D.

Use the `Add3DMesh` method for creating rectangular meshes. This method takes three values as input: the number of vertices in the M direction, the number of vertices in the N direction, and a variant array containing coordinates for all the vertices in the mesh.

Once the `PolygonMesh` is created, use the `MClose` and `NClose` properties to close the mesh.

For more information on creating meshes, see “Create Surfaces” in the *User’s Guide*.

Creating a polygon mesh

This example creates a 4×4 polygon mesh. The direction of the active viewport is then adjusted so that the three-dimensional nature of the mesh is more easily viewed.


```

Sub Ch8_Create3DMesh()
    Dim meshObj As AcadPolygonMesh
    Dim mSize, nSize, Count As Integer
    Dim points(0 To 47) As Double

    ' create the matrix of points
    points(0) = 0: points(1) = 0: points(2) = 0
    points(3) = 2: points(4) = 0: points(5) = 1
    points(6) = 4: points(7) = 0: points(8) = 0
    points(9) = 6: points(10) = 0: points(11) = 1
    points(12) = 0: points(13) = 2: points(14) = 0
    points(15) = 2: points(16) = 2: points(17) = 1
    points(18) = 4: points(19) = 2: points(20) = 0
    points(21) = 6: points(22) = 2: points(23) = 1
    points(24) = 0: points(25) = 4: points(26) = 0
    points(27) = 2: points(28) = 4: points(29) = 1
    points(30) = 4: points(31) = 4: points(32) = 0
    points(33) = 6: points(34) = 4: points(35) = 0
    points(36) = 0: points(37) = 6: points(38) = 0
    points(39) = 2: points(40) = 6: points(41) = 1
    points(42) = 4: points(43) = 6: points(44) = 0
    points(45) = 6: points(46) = 6: points(47) = 0

    mSize = 4: nSize = 4

    ' creates a 3Dmesh in model space
    Set meshObj = ThisDrawing.ModelSpace.Add3DMesh(mSize, nSize, points)

    ' Change the viewing direction of the viewport
    ' to better see the cylinder
    Dim NewDirection(0 To 2) As Double
    NewDirection(0) = -1
    NewDirection(1) = -1
    NewDirection(2) = 1
    ThisDrawing.ActiveViewport.Direction = NewDirection
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport
    ZoomAll
End Sub

```

Creating a Polyface Mesh

Use the `AddPolyfaceMesh` method to create a polyface mesh, with each face capable of having numerous vertices.

Creating a polyface mesh is similar to creating a rectangular mesh. To create a polyface mesh, specify coordinates for all its vertices then define each face by entering vertex numbers for all the vertices of that face. As you create the polyface mesh, you can set specific edges to be invisible, assign them to layers, or give them colors.

To make an edge invisible, enter the vertex number for the edge as a negative value. For more information on creating polyface meshes, see the AddPolyfaceMesh method of the *ActiveX and VBA Reference*.

For more information on creating polyface meshes, see “Create a Polyface Mesh” in the *User’s Guide*.

Creating a polyface mesh

This example creates a Polyface Mesh object in model space. The viewing direction of the active viewport is updated to display the three-dimensional nature of the mesh more easily.

```
Sub Ch8_CreatePolyfaceMesh()
    ' Define the mesh vertices
    Dim vertex(0 To 17) As Double
    vertex(0) = 4: vertex(1) = 7: vertex(2) = 0
    vertex(3) = 5: vertex(4) = 7: vertex(5) = 0
    vertex(6) = 6: vertex(7) = 7: vertex(8) = 0
    vertex(9) = 4: vertex(10) = 6: vertex(11) = 0
    vertex(12) = 5: vertex(13) = 6: vertex(14) = 0
    vertex(15) = 6: vertex(16) = 6: vertex(17) = 1

    ' Define the face list
    Dim FaceList(0 To 7) As Integer
    FaceList(0) = 1
    FaceList(1) = 2
    FaceList(2) = 5
    FaceList(3) = 4
    FaceList(4) = 2
    FaceList(5) = 3
    FaceList(6) = 6
    FaceList(7) = 5
    ' Create the polyface mesh
    Dim polyFaceMeshObj As AcadPolyfaceMesh
    Set polyFaceMeshObj = ThisDrawing.ModelSpace.AddPolyfaceMesh _
        (vertex, FaceList)

    ' Change the viewing direction of the viewport to
    ' better see the polyface mesh
    Dim NewDirection(0 To 2) As Double
    NewDirection(0) = -1
    NewDirection(1) = -1
    NewDirection(2) = 1
    ThisDrawing.ActiveViewport.direction = NewDirection
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport
    ZoomAll
End Sub
```

Creating Solids

A solid object (3DSolid object) represents the entire volume of an object. Solids are the most informationally complete and least ambiguous of the 3D modeling types. Complex solid shapes are also easier to construct and edit than wireframes and meshes.

You create solids from one of the basic solid shapes of box, cone, cylinder, sphere, torus, and wedge or by extruding a 2D object along a path or revolving a 2D object about an axis. Use one of the following methods to create solids:

AddBox, AddCone, AddCylinder, AddEllipticalCone, AddEllipticalCylinder, AddExtrudedSolid, AddExtrudedSolidAlongPath, AddRevolvedSolid, AddSolid, AddSphere, AddTorus, or AddWedge.

Like meshes, solids are displayed as wireframes until you hide, shade, or render them. Additionally, you can analyze solids for their mass properties (volume, moments of inertia, center of gravity, and so forth). Use the following properties to analyze solids: MomentOfInertia, PrincipalDirections, PrincipalMoments, ProductOfInertia, RadiiOfGyration, and Volume.

The ContourlinesPerSurface property controls the number of tessellation lines used to visualize curved portions of the wireframe. The RenderSmoothness property adjusts the smoothness of shaded and hidden-line objects.

For more information on creating solids, see “Create 3D Solids” in the *User’s Guide*.

Creating a wedge solid

The following example creates a wedge-shaped solid in model space. The viewing direction of the active viewport is updated to display the three-dimensional nature of the wedge more easily.

```

Sub Ch8_CreateWedge()
    Dim wedgeObj As Acad3DSolid
    Dim center(0 To 2) As Double
    Dim length As Double
    Dim width As Double
    Dim height As Double

    ' Define the wedge
    center(0) = 5#: center(1) = 5#: center(2) = 0
    length = 10#: width = 15#: height = 20#

    ' Create the wedge in model space
    Set wedgeObj = ThisDrawing.ModelSpace.AddWedge(center, length, width, height)

    ' Change the viewing direction of the viewport
    Dim NewDirection(0 To 2) As Double
    NewDirection(0) = -1
    NewDirection(1) = -1
    NewDirection(2) = 1
    ThisDrawing.ActiveViewport.direction = NewDirection
    ThisDrawing.ActiveViewport = ThisDrawing.ActiveViewport
    ZoomAll
End Sub

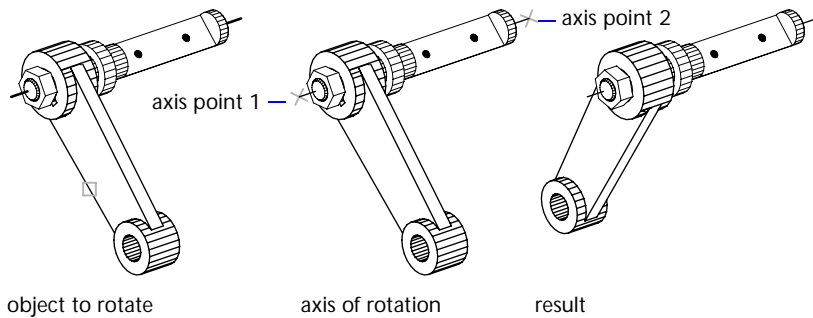
```

Editing in 3D

This section describes how to edit 3D objects by, for example, rotating, arraying, and mirroring.

Rotating in 3D

With the Rotate method, you can rotate objects in 2D about a specified point. The direction of rotation is determined by the WCS. The Rotate3D method rotates objects in 3D about a specified axis. The Rotate3D method takes three values as input: the WCS coordinates of the two points defining the rotation axis and the rotation angle in radians.



To rotate 3D objects, use either the **Rotate** or **Rotate3D** method.

For more information on rotating in 3D, see “Rotate Objects” in the *User’s Guide*.

Creating a 3D box and rotating it about an axis

This example creates a 3D box. It then defines the axis for rotation and finally rotates the box 30 degrees about the axis.

```
Sub Ch8_Rotate_3DBox()
    Dim boxObj As Acad3DSolid
    Dim length As Double
    Dim width As Double
    Dim height As Double
    Dim center(0 To 2) As Double

    ' Define the box
    center(0) = 5: center(1) = 5: center(2) = 0
    length = 5
    width = 7
    height = 10

    ' Create the box object in model space
    Set boxObj = ThisDrawing.ModelSpace.AddBox(center, length, width, height)

    ' Define the rotation axis with two points
    Dim rotatePt1(0 To 2) As Double
    Dim rotatePt2(0 To 2) As Double
    Dim rotateAngle As Double
    rotatePt1(0) = -3: rotatePt1(1) = 4: rotatePt1(2) = 0
    rotatePt2(0) = -3: rotatePt2(1) = -4: rotatePt2(2) = 0
    rotateAngle = 30
    rotateAngle = rotateAngle * 3.141592 / 180#

    ' Rotate the box
    boxObj.Rotate3D rotatePt1, rotatePt2, rotateAngle
    ZoomAll
End Sub
```

Arraying in 3D

With the `ArrayRectangular` method, you can create a rectangular array in 3D. In addition to specifying the number of columns (*X* direction) and rows (*Y* direction), you also specify the number of levels (*Z* direction).

For more information on using arrays of objects in 3D, see “Create an Array of Objects” in the *User’s Guide*.

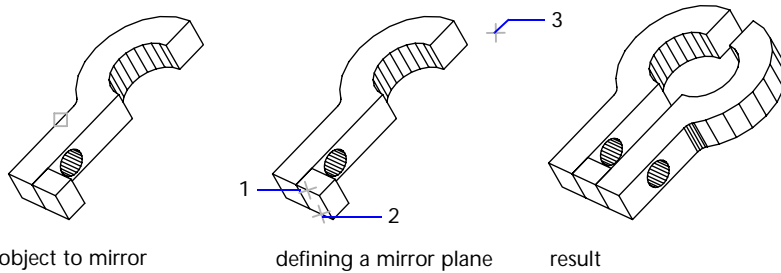
Creating a 3D rectangular array

This example creates a circle and then uses that circle to create a rectangular array of 4 rows, 4 columns, and 3 levels of circles.

```
Sub Ch8_CreateRectangularArray()  
    ' Create the circle  
    Dim circleObj As AcadCircle  
    Dim center(0 To 2) As Double  
    Dim radius As Double  
    center(0) = 2: center(1) = 2: center(2) = 0  
    radius = 0.5  
    Set circleObj = ThisDrawing.ModelSpace.AddCircle(center, radius)  
  
    ' Define the rectangular array  
    Dim numberOfRows As Long  
    Dim numberOfColumns As Long  
    Dim numberOfLevels As Long  
    Dim distanceBwtnRows As Double  
    Dim distanceBwtnColumns As Double  
    Dim distanceBwtnLevels As Double  
    numberOfRows = 4  
    numberOfColumns = 4  
    numberOfLevels = 3  
    distanceBwtnRows = 1  
    distanceBwtnColumns = 1  
    distanceBwtnLevels = 4  
  
    ' Create the array of objects  
    Dim retObj As Variant  
    retObj = circleObj.ArrayRectangular _  
        (numberOfRows, numberOfColumns, _  
        numberOfLevels, distanceBwtnRows, _  
        distanceBwtnColumns, distanceBwtnLevels)  
    ZoomAll  
End Sub
```

Mirroring in 3D

With the `Mirror3D` method, you can mirror objects along a specified mirroring plane specified by three points.



For more information on mirroring objects in 3D, see “Mirror Objects” in the *User’s Guide*.

Mirroring in 3D

This example creates a box in model space. It then mirrors the box about a plane and colors the mirrored box red.

```
Sub Ch8_MirrorABox3D()
' Create the box object
Dim boxObj As Acad3DSolid
Dim length As Double
Dim width As Double
Dim height As Double
Dim center(0 To 2) As Double
center(0) = 5#: center(1) = 5#: center(2) = 0
length = 5#: width = 7: height = 10#

' Create the box (3DSolid) object in model space
Set boxObj = ThisDrawing.ModelSpace.AddBox(center, length, width, height)

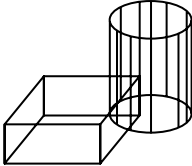
' Define the mirroring plane with three points
Dim mirrorPt1(0 To 2) As Double
Dim mirrorPt2(0 To 2) As Double
Dim mirrorPt3(0 To 2) As Double

mirrorPt1(0) = 1.25: mirrorPt1(1) = 0: mirrorPt1(2) = 0
mirrorPt2(0) = 1.25: mirrorPt2(1) = 2: mirrorPt2(2) = 0
mirrorPt3(0) = 1.25: mirrorPt3(1) = 2: mirrorPt3(2) = 2

' Mirror the box
Dim mirrorBoxObj As Acad3DSolid
Set mirrorBoxObj = boxObj.Mirror3D _
(mirrorPt1, mirrorPt2, mirrorPt3)
mirrorBoxObj.Color = acRed
ZoomAll
End Sub
```

Editing 3D Solids

Once you have created a solid, you can create more complex shapes by combining solids. You can join solids, subtract solids from each other, or find the common volume (overlapping portion) of solids. Use the Boolean or CheckInterference methods to perform these combinations.



solids before Boolean intersection



resulting solid from Boolean intersection

Solids are further modified by obtaining the 2D cross section of a solid or slicing a solid into two pieces. Use the SectionSolid method to find cross sections of solids, and the SliceSolid method for slicing a solid into two pieces.

Finding the interference between two solids

This example creates a box and a cylinder in model space. It then finds the interference between the two solids and creates a new solid from that interference. For ease of viewing, the box is colored white, the cylinder is colored cyan, and the interference solid is colored red.


```

Sub Ch8_FindInterferenceBetweenSolids()
    ' Define the box
    Dim boxObj As Acad3DSolid
    Dim length As Double
    Dim width As Double
    Dim height As Double
    Dim center(0 To 2) As Double
    center(0) = 5: center(1) = 5: center(2) = 0
    length = 5
    width = 7
    height = 10

    ' Create the box object in model space
    ' and color it white
    Set boxObj = ThisDrawing.ModelSpace. _
        AddBox(center, length, width, height)
    boxObj.Color = acWhite

    ' Define the cylinder
    Dim cylinderObj As Acad3DSolid
    Dim cylinderRadius As Double
    Dim cylinderHeight As Double
    center(0) = 0: center(1) = 0: center(2) = 0
    cylinderRadius = 5
    cylinderHeight = 20

    ' Create the Cylinder and
    ' color it cyan
    Set cylinderObj = ThisDrawing.ModelSpace.AddCylinder _
        (center, cylinderRadius, cylinderHeight)
    cylinderObj.Color = acCyan

    ' Find the interference between the two solids
    ' and create a new solid from it. Color the
    ' new solid red.
    Dim solidObj As Acad3DSolid
    Set solidObj = boxObj.CheckInterference(cylinderObj, True)
    solidObj.Color = acRed
    ZoomAll
End Sub

```

Slicing a solid into two solids

This example creates a box in model space. It then slices the box based on a plane defined by three points. The slice is returned as a 3DSolid.

```
Sub Ch8_SliceABox()  
    ' Create the box object  
    Dim boxObj As Acad3DSolid  
    Dim length As Double  
    Dim width As Double  
    Dim height As Double  
    Dim center(0 To 2) As Double  
    center(0) = 5#: center(1) = 5#: center(2) = 0  
    length = 5#: width = 7: height = 10#  
  
    ' Create the box (3DSolid) object in model space  
    Set boxObj = ThisDrawing.ModelSpace.AddBox(center, length, width, height)  
    boxObj.Color = acWhite  
  
    ' Define the section plane with three points  
    Dim slicePt1(0 To 2) As Double  
    Dim slicePt2(0 To 2) As Double  
    Dim slicePt3(0 To 2) As Double  
  
    slicePt1(0) = 1.5: slicePt1(1) = 7.5: slicePt1(2) = 0  
    slicePt2(0) = 1.5: slicePt2(1) = 7.5: slicePt2(2) = 10  
    slicePt3(0) = 8.5: slicePt3(1) = 2.5: slicePt3(2) = 10  
  
    ' slice the box and color the new solid red  
    Dim sliceObj As Acad3DSolid  
    Set sliceObj = boxObj.SliceSolid_(  
        slicePt1, slicePt2, slicePt3, True)  
    sliceObj.Color = acRed  
    ZoomAll  
End Sub
```

Defining Layouts and Plotting

9

After you've created your drawing with AutoCAD, you usually plot it on paper. A plotted drawing can contain a single view of your drawing or a more complex arrangement of views. In paper space, you can create windows called floating viewports, which display various views of the drawing. Depending on your needs, you can plot one or more viewports, or set options that determine what is plotted and how the image fits on the paper.

In this chapter

- Understanding Model Space and Paper Space
- Understanding Layouts
- Understanding Viewports
- Plotting Your Drawing

Understanding Model Space and Paper Space

Model space is the drawing environment in which you create the geometry for your model. Normally, as you begin to draw in model space, you designate your drawing limits to determine the extents of the drawing environment, and you draw in real world units.

Paper space represents the paper representation of your model as it will be plotted. In paper space you can lay out different views of your drawing, scale views independently from one another, and arrange the different views of your drawing as you want them to be plotted. There can be many different paper space representations of your drawing.

For more information about model space and paper space, see “Work in Paper Space and Model Space” in chapter 21, “Create Layouts,” in the *User’s Guide*.

Understanding Layouts

All the geometry of your drawing is contained in layouts. Model space geometry is contained on a single layout named Model. You cannot rename the model space layout, nor can you create another model space layout. There can be only one model space layout per drawing.

Paper space geometry is also contained on layouts. You can have many different paper space layouts in your drawing, each representing a different configuration to print. You can change the name of the paper space layouts.

In ActiveX Automation the ModelSpace object contains all the geometry in the model space layout. Because there can be more than one paper space layout in a drawing, the PaperSpace object points to the last active paper space layout.

For more information about working with paper space layouts, see chapter 21, “Create Layouts,” in the *User’s Guide*.

Understanding the Relationship between Layouts and Blocks

The content of any layout is distributed among two different ActiveX objects: the Layout object and the Block object. The Layout object contains the plot settings and the visual properties of the layout as it appears in the AutoCAD user interface. The Block object contains the geometry for the layout.

Each Layout object is associated with one, and only one, Block object. To access the Block object associated with a given layout, use the Block property. Conversely, each Block object is associated with one, and only one, Layout object. To access the Layout object associated with a given Block, use the Layout property for that block.

Understanding Plot Configurations

A PlotConfiguration object is similar to a Layout object, as both contain identical plot information. The difference is that a Layout object is associated with a Block object containing the geometry to plot. A PlotConfiguration object is not associated with a particular Block object. It is simply a named collection of plot settings available for use with any geometry.

Determining Layout Settings

Layout settings control the final plotted output. These settings affect the paper size, plot scale, plot area, plot origin, and the plot device name. Understanding how to use layout settings ensures the layout plots as expected. All the settings for a layout can be changed from the Layout object properties and methods.

Selecting a Paper Size and Units

The choice of paper size depends on the plotter configured for your system. Each different plotter will have a standard list of available paper sizes. You can change the paper size for a layout by using the CanonicalMediaName property.

You can also specify the units for your layout using the PaperUnits property. This property takes one of three values: acInches, acMillimeters, or acPixels. If your plotter is configured for raster output, you must specify the output size in pixels.

Adjusting the Plot Origin

The plot origin is the lower-left corner of the specified plotted area and is controlled with the `PlotOrigin` property. Typically, the plot origin is set to (0, 0). However, you can center the plot on the sheet of paper by setting the `CenterPlot` property to `TRUE`. Centering the plot alters the plot origin.

Setting the Plot Area

When you prepare to plot a layout, you can specify the plot area to determine what will be included in the plot. To specify the plot area, use the `PlotType` property. This property requires one of the following values as input:

<code>acDisplay</code>	Prints everything that is in the current model space display. This option is unavailable when plotting from a paper space layout.
<code>acExtents</code>	Prints everything that falls within the boundaries of the currently selected space.
<code>acLimits</code>	Prints everything that is in the limits of the current space.
<code>acView</code>	Prints the view named by the <code>ViewToPlot</code> property.
<code>acWindow</code>	Prints everything in the window specified by the <code>SetWindowToPlot</code> method.
<code>acLayout</code>	Prints everything that falls within the margins of the specified paper size. This option is not available when printing from model space.

When you create a new paper space layout, the default option is `acLayout`.

Setting the Plot Scale

Generally, you draw objects at their actual size. When you plot the drawing, you can either specify a precise scale or fit the image to the paper. To specify a scale, enter either a standard or custom plot scale.

To enter a standard scale, first set the `UseStandardScale` property to `TRUE`. You can then enter the desired scale using the `StandardScale` property.

To enter a custom scale, first set the `UseStandardScale` property to `FALSE`. You can then enter the custom scale using the `SetCustomScale` method.

When you are reviewing an early draft view, a precise scale is not always important. You can use the `acScaleToFit` value of the `StandardScale` property to plot the layout at the largest possible size that fits the paper.

Setting the Lineweight Scale

Lineweights can be scaled proportionately in a layout with the plot scale. Typically, lineweights specify the linewidth of plotted objects and are plotted with the linewidth size regardless of the plot scale. Most often, you use the default plot scale of 1:1 when plotting a layout. However, if you want to plot an E-size layout that is scaled to fit on an A-size sheet of paper, for example, you can specify lineweights to be scaled in proportion to the new plot scale.

To scale lineweights, set the `ScaleLineweights` property to `TRUE`. If you do not want lineweights to be scaled, set this property to `FALSE`.

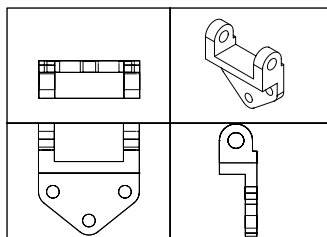
Setting the Plot Device

The plot device name is specified in the `ConfigName` property. You can set this name to any valid device name for your system. If you do not set this property, plots will be sent to the default device for your system.

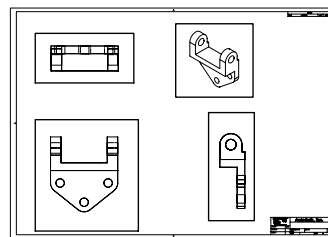
Understanding Viewports

When working in model space you draw geometry in tile viewports (referred to as Viewport objects in ActiveX Automation). You can display one or several different viewports at a time. If several tiled viewports are displayed, editing in one viewport affects all other viewports. However, you can set magnification, viewpoint, grid, and snap settings individually for each viewport.

In paper space, you work in floating paper space viewports (referred to as PViewport objects in ActiveX Automation) to contain different views of your model. Floating viewports are treated as objects you can move, resize, and shape to create a suitable layout. You also can draw objects, such as title blocks or annotations, directly in the paper space view without affecting the model itself.



tiled viewports



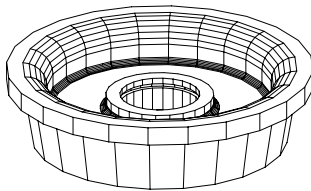
floating viewports

Refer to the *User's Guide* for more information about viewports. See “Set Model Tab Viewports” in chapter 13, “Display Multiple Views,” and “Create Layout Viewports” in chapter 21, “Create Layouts.”

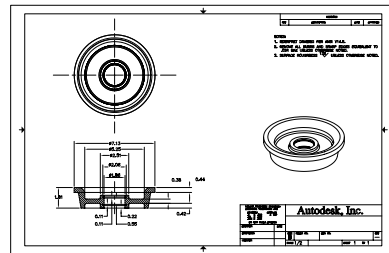
Working with Floating Viewports

You cannot edit the model in paper space. To access the model in a PViewport object, toggle from paper space to model space using the ActiveSpace property. As a result, you can work with the model while keeping the overall layout visible. In PViewport objects, the editing and view-changing capabilities are almost the same as in Viewport objects. However, you have more control over the individual views. For example, you can freeze or turn off layers in some viewports without affecting others. You can turn an entire viewport display on or off. You can also align views between viewports and scale the views relative to the overall layout.

The following illustration shows how different views of a model are displayed in paper space. Each paper space image represents a PViewport object with a different view. In one view, the dimensioning layer is frozen. Notice that the title block, border, and annotation, which are drawn in paper space, do not appear in the Model Space view. Also, the layer containing the viewport borders has been turned off.



the model



the model displayed in floating viewports

When you work in a Viewport object, the ActiveSpace property must always be set to acModel Space. When you are working in a PViewport object, you can set the ActiveSpace property to either acModel Space or acPaperSpace, thus allowing you to switch between paper space and model space as needed.

PViewport object, Viewport object, and ActiveSpace property settings

Type of viewport	Status	Usage
PViewport	ActiveSpace = acPaperspace	Arrange the layout by creating floating viewports and adding title block, borders, and annotation. Editing does not affect the model.
PViewport	ActiveSpace = acModel space	Work within floating viewports to edit the model or change views. You can turn off or freeze layers in individual viewports.
Viewport	ActiveSpace = acModel space	Split the screen into tiled viewports to edit different views of the model.

In AutoCAD ActiveX Automation, the ActiveSpace property is used to control the TILEMODE system variable. Setting Th sDrawi ng. Acti veSpace = acModel Space is equivalent to setting TILEMODE = on, and setting Th sDrawi ng. Acti veSpace = acPaperSpace is equivalent to setting TILEMODE = off.

Similarly, the MSpace property is the equivalent of both the MSPACE and PSPACE commands in AutoCAD. Setting Th sDrawi ng. MSpace = TRUE is the same as using the MSPACE command: it switches to model space. Setting Th sDrawi ng. MSpace = FALSE is the same as using the PSPACE command: it switches to paper space.

In addition, you are required to use the Display method before setting the MSpace property to TRUE. The Display method initializes certain graphic settings that must be set before switching to model space. In AutoCAD this is done “behind the scenes.” However, in the ActiveX Automation interface, the programmer must take care of this initialization.

NOTE Remember, you must turn on the display using the Display method for at least one PViewport object before you can set the MSpace property to TRUE. Failure to turn on the display will result in an error being returned when you try to set the MSpace property.

Switching to a Paper Space Layout

From model space, you can switch to the last active paper space layout.

To switch to the last active paper space layout

- 1 Set the **ActiveSpace** property to **acPaperSpace**:

```
ThisDrawing.ActiveSpace = acPaperSpace
```

- 2 Toggle the **MSpace** property to **FALSE**:

```
ThisDrawing.MSpace = FALSE
```

When you are in paper space, AutoCAD displays the paper space UCS icon in the lower-left corner of the graphics area. The crosshairs indicate that the paper space layout area (not the views in the viewports) can be edited.

Switching to the Model Space Layout

From paper space, you can switch to model space floating viewports or model space tiled viewports.

To switch to floating viewports

- 1 Use the **Display** method to initialize graphic settings:

```
ThisDrawing.ActiveViewport.Display = TRUE
```

- 2 Toggle the **MSpace** property to **TRUE**:

```
ThisDrawing.MSpace = TRUE
```

This will place you in model space, floating viewports.

NOTE You must create floating viewports before you attempt to switch to model space.

To switch to tiled viewports

To switch to tiled viewports, perform this additional step:

- Set the **ActiveSpace** property to **acModelSpace**:

```
ThisDrawing.ActiveSpace = acModelSpace
```

Creating Paper Space Viewports

Paper space viewports are created with the `AddPViewport` method. This method requires a center point and the width and height of the new viewport. Before creating the viewport, use the `ActiveSpace` property to set paper space as the current space (normally done by setting `TILEMODE` to 0).

After creating the `PViewport` object, you can set properties of the view itself, such as viewing direction (`Direction` property), lens length for perspective views (`LensLength` property), and grid display (`GridOn` property). You can also control properties of the viewport itself, such as layer (`Layer` property), linetype (`Linetype` property), and linetype scaling (`LinetypeScale` property).

Creating and enabling a floating viewport

This example switches AutoCAD to paper space, creates a floating viewport, sets the view, and enables the viewport.

```
Sub Ch9_SwitchToPaperSpace()  
    ' Set the active space to paper space  
    ThisSDrawing.ActiveSpace = acPaperSpace  
  
    ' Create the paperspace viewport  
    Dim newVport As AcadPViewport  
    Dim center(0 To 2) As Double  
    center(0) = 3.25  
    center(1) = 3  
    center(2) = 0  
    Set newVport = ThisSDrawing.PaperSpace._  
        AddPViewport(center, 6, 5)  
  
    ' Change the view direction for the viewport  
    Dim viewDir(0 To 2) As Double  
    viewDir(0) = 1  
    viewDir(1) = 1  
    viewDir(2) = 1  
    newVport.direction = viewDir  
  
    ' Enable the viewport  
    newVport.Display = True  
  
    ' Switch to model space  
    ThisSDrawing.MSpace = True  
  
    ' Set newVport current  
    ' (not always necessary but a good idea)  
    ThisSDrawing.ActivePViewport = newVport  
  
    ' Zoom Extents in model space  
    ZoomExtents
```

```

' Turn model space editing off
ThisDrawing.MSpace = False

' ZoomExtents in paperspace
ZoomExtents
End Sub

```

The order of steps in the preceding code is important. In general, things must be done in the same order they would be done at the AutoCAD command line. The only unexpected actions involve defining the view and enabling the viewport.

NOTE To set or modify aspects of the view (view direction, lens length, and so forth), the Viewport object's Display method must be set to off (FALSE), and before you can set a viewport current the Display method must be set to on (TRUE).

Creating four floating viewports

This example takes the example from “Creating and enabling a floating viewport” on page 271 and continues it by creating four floating viewports and setting the view of each to top, front, right, and isometric views, respectively. Each view is scaled to half the scale of paper space. To ensure there is something to see in these viewports, you may want to create a 3D solid sphere before trying this example.

```

Sub Ch9_FourPViewports()
    Dim topVport, frontVport As AcadPViewport
    Dim rightVport, isoVport As AcadPViewport
    Dim pt(0 To 2) As Double
    Dim viewDir(0 To 2) As Double
    ThisDrawing.ActiveSpace = acPaperSpace
    ThisDrawing.MSpace = True
    ' Take the existing PViewport and make it the topVport
    pt(0) = 2.5: pt(1) = 5.5: pt(2) = 0
    Set topVport = ThisDrawing.ActivePViewport
    ' No need to set Direction for top view
    topVport.Center = pt
    topVport.Width = 2.5
    topVport.Height = 2.5
    topVport.Display = True
    ThisDrawing.MSpace = True
    ThisDrawing.ActivePViewport = topVport
    ZoomExtents
    ZoomScaled 0.5, acZoomScaledRelativePSpace
End Sub

```

```

' Create and setup frontVport
  pt(0) = 2.5: pt(1) = 2.5: pt(2) = 0
  Set frontVport = ThisDrawing.PaperSpace. _
    AddPViewport(pt, 2.5, 2.5)
  viewDir(0) = 0: viewDir(1) = 1: viewDir(2) = 0
  frontVport.direction = viewDir
  frontVport.Display acOn
  ThisDrawing.MSpace = True
  ThisDrawing.ActivePViewport = frontVport
  ZoomExtents
  ZoomScaled 0.5, acZoomScaledRelativePSpace
' Create and setup rightVport
  pt(0) = 5.5: pt(1) = 5.5: pt(2) = 0
  Set rightVport = ThisDrawing.PaperSpace. _
    AddPViewport(pt, 2.5, 2.5)
  viewDir(0) = 1: viewDir(1) = 0: viewDir(2) = 0
  rightVport.direction = viewDir
  rightVport.Display acOn
  ThisDrawing.MSpace = True
  ThisDrawing.ActivePViewport = rightVport
  ZoomExtents
  ZoomScaled 0.5, acZoomScaledRelativePSpace
' Create and set up isoVport
  pt(0) = 5.5: pt(1) = 2.5: pt(2) = 0
  Set isoVport = ThisDrawing.PaperSpace. _
    AddPViewport(pt, 2.5, 2.5)
  viewDir(0) = 1: viewDir(1) = 1: viewDir(2) = 1
  isoVport.direction = viewDir
  isoVport.Display acOn
  ThisDrawing.MSpace = True
  ThisDrawing.ActivePViewport = isoVport
  ZoomExtents
  ZoomScaled 0.5, acZoomScaledRelativePSpace
' Finish: Perform a regen in all viewports
  ThisDrawing.Regen True
End Sub

```

Changing Viewport Views and Content

To change the view within a Viewport object, you must be in model space and the viewport must be active.

To edit a drawing in a floating viewport

- 1 In model space, make the viewport active by setting the ActiveViewport property:

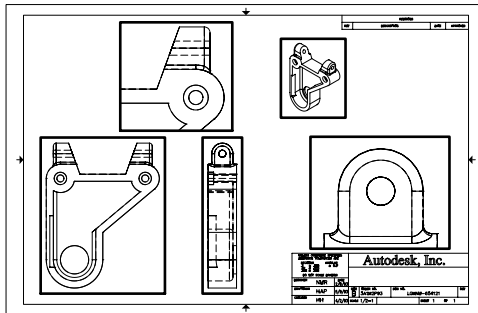
```
ThisDrawing.ActiveViewport = MyViewportObject
```

- 2 Edit the drawing.

You can also create objects such as annotations, dimensions, and title blocks in paper space. You must, however, set the `ActiveSpace` property to `FALSE`, and turn paper space on using the `MSpace` property. Objects created in paper space are visible only in paper space.

Scaling Views Relative to Paper Space

Before you plot, you can establish accurate zoom scale factors for each section of your drawing. Scaling views relative to paper space establishes a consistent scale for each displayed view. For example, the following illustration shows a paper space view with several viewports—each set to different scales and views. To scale the plotted drawing accurately, you must scale each view relative to paper space, not relative to the previous view or to the full-scale model.



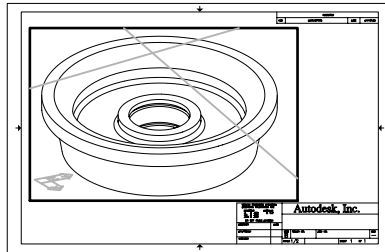
When you work in paper space, the scale factor represents a ratio between the size of the plotted drawing and the actual size of the model displayed in the viewports. To derive this scale, divide paper space units by model space units. For a quarter-scale drawing, for example, you specify a scale factor of one paper space unit to four model space units (1:4).

Use the `ZoomScaled` method to scale viewports relative to paper space units. This method takes three values as input: the viewport to scale, the scale factor, and how you want that scale factor applied. The third value is optional and determines how the scale is applied:

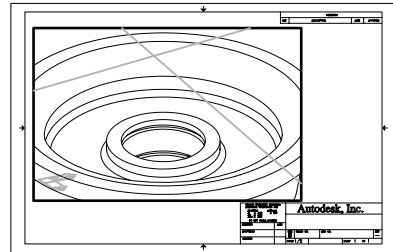
- Relative to the drawing limits
- Relative to the current view
- Relative to paper space units

To specify the scale relative to paper space units, enter the `acZoomScaled-RelativeToPaperSpace` constant for this value.

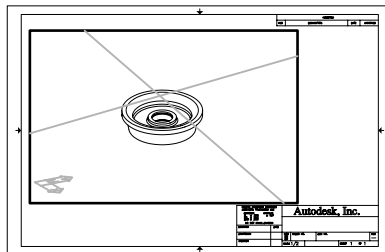
As shown in the illustrations, if you enter a scale of 2 relative to the paper space units, the scale in the viewport increases to twice the size of the paper space units. A scale of .5 relative to the paper space units sets the scale to half the size of the paper space units. The model is plotted at half its actual size.



current view



zoom to 2xp

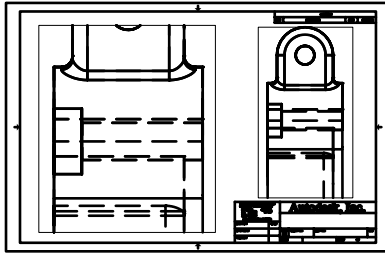


zoom to 0.5xp

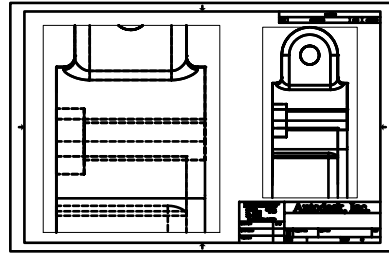
Scaling Pattern Linetypes in Paper Space

In paper space, you can scale any type of linetype in two ways. The scale can be based on the drawing units of the space in which the object was created (model or paper). The linetype scale also can be a uniform scale based on paper space units. You can use the PSLTSCALE system variable to maintain the same linetype scaling for objects displayed at different zoom scales in different viewports. It also affects the line display in 3D views.

In the following illustration, the pattern linetype of the lines in model space is scaled uniformly in paper space by the PSLTSCALE system variable. Notice that the linetype in the two viewports has the same scale, even though the objects have different zoom scales.



psltscale=1, dashes scaled to paper space



psltscale=0, dashes scaled to space where they were created

Use the `SetVariable` method to set the value of the `PSLTSCALE` system variable.

Hiding Lines in Plotted Viewports

If your drawing contains 3D faces, meshes, extruded objects, surfaces, or solids, you can remove hidden lines from specific viewports when you plot the viewport.

To hide lines in plots of paper space viewports (PViewport objects) use the `RemoveHiddenLines` property for the given viewport. This property takes a Boolean value. Enter `TRUE` to remove the hidden lines from the plot, enter `FALSE` to have the hidden lines drawn in the plot.

To hide lines in plots of model space viewports (Viewport objects) use the `PlotHidden` property found on the Layout object. This property takes a Boolean value. Enter `TRUE` to remove the hidden lines from the plot, enter `FALSE` to have the hidden lines drawn in the plot.

Plotting Your Drawing

You can plot your drawing as it is viewed in model space, or you can plot one of your prepared paper space layouts. Plotting from model space is often preferable when you want to view or verify your drawing prior to creating a paper space layout. Once your model is ready, you can prepare and plot a paper space layout.

Plotting involves working with two ActiveX Automation objects: the Layout object and the Plot object. The Layout object contains the plot settings for a given layout. The Plot object contains the methods and properties that initiate and monitor a plotting sequence.

For more information about plotting, see chapter 22, “Plot Drawings,” in the *User’s Guide*.

Performing Basic Plotting

From the Plot object you can use the following methods and properties:

<code>PlotToFile</code>	Plots to a file
<code>PlotToDevice</code>	Plots to a plotter or printer
<code>DisplayPlotPreview</code>	Displays a preview of the specified plot
<code>SetLayoutsToPlot</code>	Establish the list of layouts that will be plotted in the next call to <code>PlotToFile</code> or <code>PlotToDevice</code>
<code>StartBatchMode</code>	Initiates a batch plot
<code>QuietErrorMode</code>	Toggles the quiet error mode for plot error reporting
<code>NumberOfCopies</code>	Specifies the number of copies to be plotted
<code>BatchPlotProgress</code>	Gets the current status of the batch plot, or terminates the batch plot.

The `SetLayoutsToPlot` method must be called before each `PlotToDevice` or `PlotToFile` method. If `SetLayoutsToPlot` is not called, or is called with a `NULL` input, the active layout will be plotted.

The `NumberOfCopies` property specifies the number of copies to plot. If this property is not reset before each `PlotToDevice` call, the last value specified in the `NumberOfCopies` property will be used.

Batch mode plotting is provided to support the `BatchPlot` utility application. Before initiating a batch plot, set the `QuietErrorMode` to `TRUE` to have an uninterrupted plot session. Use the `StartBatchMode` method to initiate a batch plot. Use the `BatchPlotProgress` property to check on the progress of the batch plot, or to terminate the batch mode.

Plotting from Model Space

Typically, when you plot a large drawing such as a floorplan, you can specify a scale to convert the real drawing units into plotted inches or millimeters. However, when you plot from model space, the defaults that are used if there are no settings specified include plot to system printer, plot the current display, scaled to fit, 0 rotation, and 0,0 offset. To modify the plot settings, change the properties on the Layout object associated with model space.

Plotting the extents of an active model space layout

This example first checks to make sure the active space is model space. The example then establishes several plot settings. Finally, the plot is sent using the PlotToDevice method.

```
Sub Ch9_PrintModel Space()  
    ' Verify that the active space is model space  
    If ThisDrawing.ActiveSpace = acPaperSpace Then  
        ThisDrawing.MSpace = True  
        ThisDrawing.ActiveSpace = acModel Space  
    End If  
  
    ' Set the extents and scale of the plot area  
    ThisDrawing.Model Space.Layout.PlotType = acExtents  
    ThisDrawing.Model Space.Layout._  
        StandardScale = acScaleToFit  
  
    ' Set the number of copies to one  
    ThisDrawing.Plot.NumberOfCopies = 1  
  
    ' Initiate the plot  
    ThisDrawing.Plot.PlotToDevice  
End Sub
```

The device name can be specified using the ConfigName property. This device can be overridden in the PlotToDevice method by specifying a PC3 file.

Plotting from Paper Space

You can plot one or many paper space layouts at a time. You can plot the active layout, as demonstrated in “Plotting from Model Space” on page 278, or you can specify by name the layouts to be plotted.

Plotting two paper space layouts

This example sends the two paper space layouts “Layout1” and “Layout2” to the default plot device. Note that these two layouts must exist in the drawing for this code to run. This example first creates a string array containing the names of the layouts to be plotted. It then uses this array as input to the SetLayoutsToPlot method. The example then sets the number of copies to be plotted, and finally sends the plot to the default device.

```
Sub Ch9_PrintPaperSpace()  
    ' Establish the paper space layouts to be plotted  
    Dim strLayouts(0 To 1) As String  
    Dim varLayouts As Variant  
    strLayouts(0) = "Layout1"  
    strLayouts(1) = "Layout2"  
    varLayouts = strLayouts  
    ThisDrawing.Plot.SetLayoutsToPlot varLayouts  
  
    ' Set the number of copies to one  
    ThisDrawing.Plot.NumberOfCopies = 1  
  
    ' Initiate the plot  
    ThisDrawing.Plot.PlotToDevice  
End Sub
```


Advanced Drawing and Organizational Techniques

As you gain experience, you can take advantage of the many advanced features of AutoCAD to further enhance your applications.

You can include in your drawing raster images such as aerial, satellite, and digital photographs, as well as computer-rendered images.

In addition to enhancing your drawing's visual image, AutoCAD provides several features to help you organize data, allowing you to further expand the intelligence of the objects in your drawing.

10

In this chapter

- Working with Raster Images
- Using Blocks and Attributes
- Using External References
- Assigning and Retrieving Extended Data

Working with Raster Images

With AutoCAD you can add raster images to your vector-based AutoCAD drawings, then view and plot the resulting file.

Attaching and Scaling a Raster Image

Images can be placed in a drawing file, but they are not actually part of the file. The image is linked to the drawing file through a path name or a data management document ID. Linked image paths can be changed or removed at any time. To attach an image, you create a Raster object in your drawing using the `AddRaster` method. This method takes four values as input: the name of the image file to attach, the insertion point in the drawing to place the image, the scale factor of the image, and the rotation angle of the image. Remember, the Raster object represents an independent link to the image, not the image itself.

Once you've attached an image, you can reattach it many times, creating a new Raster object for each attachment. Each attachment has its own clip boundary and its own settings for brightness, contrast, fade, and transparency. A single image can be cut into multiple pieces and rearranged independently in your drawing.

You can set the raster image scale factor when you create the Raster object so that the image's geometry scale matches the scale of the geometry created in the AutoCAD drawing. When you select an image to attach, the image is inserted at a scale factor of 1 image unit of measurement to 1 AutoCAD unit of measurement. To set the image scale factor, you need to know the scale of the geometry on the image, and you need to know what unit of measurement (inches, feet, and so forth) you want to use to define 1 AutoCAD unit. The image file must contain resolution information defining the DPI, or dots per inch, and number of pixels in the image.

If an image has resolution information, AutoCAD combines it with the scale factor and the AutoCAD unit of measurement you supply to scale the image in your drawing. For example, if your raster image is a scanned blueprint on which the scale is 1 inch equals 50 feet, or 1:600, and your AutoCAD drawing is set up so that 1 unit represents 1 inch, then to set the scale factor of the image, you enter 600 for the `ScaleFactor` parameter of the `AddRaster` method. AutoCAD then inserts the image at a scale that brings the geometry in the image into alignment with the vector geometry in the drawing.

NOTE If no resolution information is defined with the attached image file, AutoCAD calculates the image's original width as one unit. After insertion, the image width in AutoCAD units is equal to the scale factor.

For more information about attaching and scaling raster images, see “Attach, Scale, and Detach Raster Images” in chapter 29, “Work with Raster Images in AutoCAD Drawings,” in the *User's Guide*.

Attaching a raster image

This example adds a raster image in model space. This example uses the *watch.jpg* found in the sample directory. If you do not have this image, or it is located in a different directory, insert a valid path and file name for the `imageName` variable.

```
Sub Ch10_AttachingARaster()  
    Dim insertionPoint(0 To 2) As Double  
    Dim scaleFactor As Double  
    Dim rotationAngle As Double  
    Dim imageName As String  
    Dim rasterObj As AcadRasterImage  
    imageName = "C:/Program Files/AutoCAD 2002/sample/watch.jpg"  
    insertionPoint(0) = 5  
    insertionPoint(1) = 5  
    insertionPoint(2) = 0  
    scaleFactor = 2  
    rotationAngle = 0  
  
    On Error GoTo ERRORHANDLER  
    ' Attach the raster image in model space  
    Set rasterObj = ThisDrawing.ModelSpace.AddRaster _  
        (imageName, insertionPoint, _  
        scaleFactor, rotationAngle)  
  
    ZoomAll  
    Exit Sub  
ERRORHANDLER:  
    MsgBox Err.Description  
End Sub
```

Managing Raster Images

You can manage raster image name, file name, and file path using the properties of the Raster object.

For information about managing raster images, see “Manage Raster Images” in chapter 29, “Work with Raster Images in AutoCAD Drawings,” in the *User's Guide*.

Changing Image File Paths

The path and file name of an image is queried or changed using the ImageFile property. The path set by this property is the actual path where AutoCAD looks for the image.

If AutoCAD cannot locate the drawing (for example, if you have moved the file to a different directory than the one saved with the ImageFile property), it removes relative or absolute path information from the name (for example, `\images\tree.tga` or `c:\my project\images\tree.tga` becomes `tree.tga`) and searches the paths you have defined using the SetProjectFilePath method on the Preferences object. If the drawing is not located in the paths, it attempts the first search path again.

You can remove the path from the file name or specify a relative path by resetting the ImageFile property.

Changing the path in the ImageFile property does not affect the project files' search-path settings.

Naming Images

Image names are not necessarily the same as image file names. When you attach an image to a drawing, AutoCAD uses the file name without the file extension as the image name. You can change the image name without affecting the name of the file.

The image file is represented by the ImageFile property on the Raster object. Changing the ImageFile property will change the image in the drawing. The image name is represented by the Name property, and changing the Name property will change the name of the image only, not the file associated with it.

Modifying Images and Image Boundaries

All images have an image boundary. When you attach an image to a drawing, the image boundary inherits the current property settings, including color, layer, linetype, and linetype scale. If the image is a bitonal image, the image color and boundary color are the same.

As with other AutoCAD objects, you can modify images and their boundary properties. For example, you can:

- Display or hide the image boundary
- Modify the image layer, boundary color, and linetype

- Change the image location
- Scale, rotate, and change the width and height of the image
- Toggle the image visibility
- Change the image transparency
- Change the image brightness, contrast, and fade
- Change the quality and speed of image display

For more information about working with image boundaries, see “Modify Raster Images and Image Boundaries” in chapter 29, “Work with Raster Images in AutoCAD Drawings,” in the *User’s Guide*.

Showing and Hiding Image Boundaries

Hiding an image boundary ensures that the image cannot accidentally be moved or modified and prevents the boundary from being plotted or displayed. When image boundaries are hidden, clipped images are still displayed to their specified boundary limits; only the boundary is affected. Showing and hiding image boundaries affects all images attached to your drawing.

To show or hide the image boundaries, use the `ClippingEnabled` property.

NOTE This property affects only the image boundary. To see a change in the image when toggling this property, look closely at the small boundary surrounding the image.

For more information on this topic, see “Show and Hide Raster Image Boundaries” in chapter 29, “Work with Raster Images in AutoCAD Drawings,” in the *User’s Guide*.

Changing Image Layer, Boundary Color, and Boundary Linetype

You can change the color and linetype of image boundaries and the layer of an image using the following properties:

Layer	Specifies the layer for the image
Color	Specifies the color of the image boundary
Linetype	Specifies the linetype of the image

For more information on this topic, see “Modify Raster Images and Image Boundaries” in chapter 29, “Work with Raster Images in AutoCAD Drawings,” in the *User’s Guide*.

Changing Image Scale, Rotation, Location, Width, and Height

You can change the scale, rotation, location, width, and height of an image using the following methods and properties:

ScaleEntity	Scales the image
Rotate	Rotates the image
Origin	Specifies the image location
Width	Specifies the width of the image in pixels
Height	Specifies the height of the image in pixels
ImageWidth	Specifies the width of the image in database units
ImageHeight	Specifies the height of the image in database units
ShowRotation	Determines if the raster is displayed rotated

For more information on this topic, see “Scale Raster Images” in chapter 29, “Work with Raster Images in AutoCAD Drawings,” in the *User’s Guide*.

Changing Image Visibility

Image visibility affects the redraw speed by hiding images in the current drawing session. Hidden images are not displayed or plotted; only the drawing boundary is displayed. To hide images, set the ImageVisibility property to FALSE. To redisplay the images, set the ImageVisibility property to TRUE.

Modifying Bitonal Image Color and Transparency

Bitonal raster images are images consisting only of a foreground color and a background color. When you attach a bitonal image, the foreground pixels in the image inherit the current layer settings for color. In addition to the modifications you can make to any attached image, you can modify bitonal images by changing the foreground color and by turning the transparency of the background on and off.

NOTE Bitonal images and bitonal image boundaries are always the same color.

To change the foreground color of a bitonal image, use the Color property. To turn the transparency on and off, use the Transparency property.

For more information on this topic, see “Modify Bitonal Raster Image Color and Transparency” in chapter 29, “Work with Raster Images in AutoCAD Drawings,” in the *User’s Guide*.

Adjusting Image Brightness, Contrast, and Fade

You can adjust image brightness, contrast, and fade in AutoCAD to the display of the image and to plotted output without affecting the original raster image file.

Use the following properties to adjust brightness, contrast, and fade:

Brightness	Specifies the brightness level of an image
Contrast	Specifies the contrast level of an image
Fade	Specifies the fade level of an image

For more information on this topic, see “Change Raster Image Brightness, Contrast, and Fade” in chapter 29, “Work with Raster Images in AutoCAD Drawings,” in the *User’s Guide*.

Clipping Images

You can define a region of an image for display and plotting by clipping the image. The clipping boundary must be a 2D polygon or rectangle with vertices constrained to lay within the boundaries of the image. Multiple instances of the same image can have different boundaries.

For more information about clipping images, see “Clip Raster Images” in chapter 29, “Work with Raster Images in AutoCAD Drawings,” in the *User’s Guide*.

To clip an image

- 1 Turn on the image boundaries using the `ClippingEnabled` property.
- 2 Specify the clipping boundary and perform the clip using the `ClipBoundary` method. This method takes one value as input: a variant array of 2D WCS coordinates specifying the clipping boundary of a raster image.

Changing the Clipping Boundary

To change an existing clipping boundary, simply repeat the previous steps. The old boundary will be deleted and the new boundary will replace the old one.

Showing and Hiding the Clipping Boundary

You can display a clipped image using the clipping boundary, or you can hide the clipping boundary and display the original image boundaries. To hide a clipping boundary and display the original image, set the `ClippingEnabled` property to `FALSE`. To display the clipped image, set the `ClippingEnabled` property to `TRUE`.

Clipping a raster image boundary

This example adds a raster image in model space. It then clips the image based on a clip boundary. This example uses *downtown.jpg* found in the sample directory. If you do not have this image, or it is located in a different directory, insert a valid path and file name for the `imageName` variable.

```
Sub Ch10_ClippingRasterBoundary()
    Dim insertPoint(0 To 2) As Double
    Dim scaleFactor As Double
    Dim rotationAngle As Double
    Dim imageName As String
    Dim rasterObj As AcadRasterImage

    imageName = "C:\AutoCAD\sample\downtown.jpg"
    insertPoint(0) = 5
    insertPoint(1) = 5
    insertPoint(2) = 0
    scaleFactor = 2
    rotationAngle = 0

    On Error GoTo ERRORHANDLER
    ' Creates a raster image in model space
    Set rasterObj = ThisDrawing.ModelSpace.AddRaster _
        (imageName, insertPoint, _
        scaleFactor, rotationAngle)

    ZoomAll

    ' Establish the clip boundary with an array of points
    Dim clipPoints(0 To 9) As Double
    clipPoints(0) = 6: clipPoints(1) = 6.75
    clipPoints(2) = 7: clipPoints(3) = 6
    clipPoints(4) = 6: clipPoints(5) = 5
    clipPoints(6) = 5: clipPoints(7) = 6
    clipPoints(8) = 6: clipPoints(9) = 6.75

    ' Clip the image
    rasterObj.ClipBoundary clipPoints

    ' Enable the display of the clip
    rasterObj.ClippingEnabled = True
    ThisDrawing.Regen acActiveViewport
    Exit Sub

ERRORHANDLER:
    MsgBox Err.Description
End Sub
```

Using Blocks and Attributes

AutoCAD provides several features to help you manage objects in your drawings. With blocks you can organize and manipulate many objects as one component. Attributes associate items of information with the blocks in your drawings—for example, part numbers and prices.

Using AutoCAD external references, or xrefs, you can attach or overlay entire drawings to your current drawing. When you open your current drawing, any changes made in the referenced drawing appear in the current drawing.

Working with Blocks

A block is a collection of objects you can associate together to form a single object, or a block reference. You can insert, scale, and rotate a block reference in a drawing. You can explode a block reference into its component objects, modify them, and redefine the block. AutoCAD updates all future instances of that block reference based on the definition of the block.

Blocks can be defined from objects originally drawn on different layers with different colors and linetypes. You can preserve the layer, color, and linetype information of objects in a block. Then, each time you insert the block, you have each object within the block drawn on its original layer with its original color and linetype.

For more information about working with blocks, see “Create and Insert Symbols (Blocks)” in chapter 16, “Draw Geometric Objects,” in the *User’s Guide*.

Defining Blocks

To create a new block, use the Add method. This method requires two values as input: the location in the drawing where the block is added and the name of the block to create.

Once created, you can add any geometrical object, or another block, to the newly created block. You can then insert an instance of the block into the drawing. An inserted block is an object called a block reference.

You can also create a block by using the WBlock method to group objects in a separate drawing file. The drawing file can then be used as a block definition for other drawings. AutoCAD considers any drawing you insert into another drawing to be a block.

For more information on defining blocks, see “Create Blocks” in chapter 16, “Draw Geometric Objects,” in the *User’s Guide*.

Inserting Blocks

You can insert blocks or entire drawings into the current drawing with the InsertBlock method. The InsertBlock method takes six values as input: the insertion point, the name of the block or drawing to insert, the X-scale factor, the Y-scale factor, the Z-scale factor, and the rotation angle.

When you insert an entire drawing into another drawing, AutoCAD treats the inserted drawing like any other block reference. Subsequent insertions reference the block definition (which contains the geometric description of the block) with different position, scale, and rotation settings. If you change the original drawing after inserting it, the changes have no effect on the inserted block. If you want the inserted block to reflect the changes you made to the original drawing, you can redefine the block by reinserting the original drawing. This can be done with the InsertBlock method.

If you insert a drawing as a block, the file name is automatically used as the name of the block. You can change the name of the block by using the Name property once the block has been created.

By default, AutoCAD uses the coordinate (0, 0, 0) as the base point for inserted drawings. You can change the base point of a drawing by opening the original drawing and using the SetVariable method to specify a different insertion base point for the INSBASE system variable. AutoCAD uses the new base point the next time you insert the drawing.

If the drawing you insert contains PaperSpace objects, those objects are not included in the current drawing’s block definition. To use the PaperSpace objects in another drawing, open the original drawing and use the Add method to define the PaperSpace objects as a block. You can insert the drawing into another drawing in either paper space or model space.

A block reference cannot be iterated to find the original objects that compose it. However, you can iterate the original block definition, or you can explode the block reference into its original components.

You can also insert an array of blocks using the AddMInsertBlock method. This method does not insert a single block into your drawing, as the InsertBlock does, but instead inserts an array of the specified block. This method returns an MInsertBlock object.

For more information on inserting blocks, see “Insert Blocks” in chapter 16, “Draw Geometric Objects,” in the *User’s Guide*.

Defining a block and inserting the block into a drawing

This example defines a block and adds a circle to the block definition. It then inserts the block into the drawing as a block reference.

```
Sub Ch10_InsertingABlock()  
    ' Define the block  
    Dim blockObj As AcadBlock  
    Dim insertionPnt(0 To 2) As Double  
    insertionPnt(0) = 0  
    insertionPnt(1) = 0  
    insertionPnt(2) = 0  
    Set blockObj = ThisDrawing.Blocks.Add _  
        (insertionPnt, "CircleBlock")  
  
    ' Add a circle to the block  
    Dim circleObj As AcadCircle  
    Dim center(0 To 2) As Double  
    Dim radius As Double  
    center(0) = 0  
    center(1) = 0  
    center(2) = 0  
    radius = 1  
    Set circleObj = blockObj.AddCircle(center, radius)  
  
    ' Insert the block  
    Dim blockRefObj As AcadBlockReference  
    insertionPnt(0) = 2  
    insertionPnt(1) = 2  
    insertionPnt(2) = 0  
    Set blockRefObj = ThisDrawing.ModelSpace.InsertBlock _  
        (insertionPnt, "CircleBlock", 1#, 1#, 1#, 0)  
    ZoomAll  
    MsgBox "The circle belongs to " & blockRefObj.ObjectName  
End Sub
```

NOTE After insertion, the external file’s WCS is aligned parallel to the XY plane of the current UCS in the current drawing. Thus, a block from an external file is inserted at any orientation in space by setting the UCS before inserting it.

Exploding a Block Reference

Use the Explode method to break a block reference. By exploding a block reference, you can modify the block or add to or delete the objects that define it.

Displaying the results of an exploded block reference

This example creates a block and adds a circle to the definition of the block. The block is then inserted into the drawing as a block reference. The block reference is then exploded, and the objects resulting from the explode process are displayed along with their object types.

```
Sub Ch10_ExplodingABlock()  
    ' Define the block  
    Dim blockObj As AcadBlock  
    Dim insertionPnt(0 To 2) As Double  
    insertionPnt(0) = 0  
    insertionPnt(1) = 0  
    insertionPnt(2) = 0  
    Set blockObj = ThisDrawing.Blocks.Add _  
        (insertionPnt, "CircleBlock")  
  
    ' Add a circle to the block  
    Dim circleObj As AcadCircle  
    Dim center(0 To 2) As Double  
    Dim radius As Double  
    center(0) = 0  
    center(1) = 0  
    center(2) = 0  
    radius = 1  
    Set circleObj = blockObj.AddCircle(center, radius)  
  
    ' Insert the block  
    Dim blockRefObj As AcadBlockReference  
    insertionPnt(0) = 2  
    insertionPnt(1) = 2  
    insertionPnt(2) = 0  
    Set blockRefObj = ThisDrawing.ModelSpace.InsertBlock _  
        (insertionPnt, "CircleBlock", 1#, 1#, 1#, 0)  
    ZoomAll  
    MsgBox "The circle belongs to " & blockRefObj.ObjectName  
  
    ' Explode the block reference  
    Dim explodedObjects As Variant  
    explodedObjects = blockRefObj.Explode  
  
    ' Loop through the exploded objects  
    Dim I As Integer  
    For I = 0 To UBound(explodedObjects)  
        explodedObjects(I).Color = acRed  
        explodedObjects(I).Update  
        MsgBox "Exploded Object " & I & ": " & _  
            explodedObjects(I).ObjectName  
        explodedObjects(I).Color = acByLayer  
        explodedObjects(I).Update  
    Next  
End Sub
```


Redefining a Block

Use any of the Block object methods and properties to redefine a block. When you redefine a block, all the references to that block in the drawing are immediately updated to reflect the new definition.

Redefinition affects previous and future insertions of a block. Constant attributes are lost and replaced by any new constant attributes. Variable attributes remain unchanged, even if the new block has no attributes.

Redefining the objects in a block definition

This example creates a block and adds a circle to the definition of the block. The block is then inserted into the drawing as a block reference. The circle in the block definition is updated, and the block reference updates automatically.

```
Sub Ch10_RedefiningABlock()  
    ' Define the block  
    Dim blockObj As AcadBlock  
    Dim insertionPnt(0 To 2) As Double  
    insertionPnt(0) = 0  
    insertionPnt(1) = 0  
    insertionPnt(2) = 0  
    Set blockObj = ThisDrawing.Blocks.Add _  
        (insertionPnt, "CircleBlock")  
  
    ' Add a circle to the block  
    Dim circleObj As AcadCircle  
    Dim center(0 To 2) As Double  
    Dim radius As Double  
    center(0) = 0  
    center(1) = 0  
    center(2) = 0  
    radius = 1  
    Set circleObj = blockObj.AddCircle(center, radius)  
  
    ' Insert the block  
    Dim blockRefObj As AcadBlockReference  
    insertionPnt(0) = 2  
    insertionPnt(1) = 2  
    insertionPnt(2) = 0  
    Set blockRefObj = ThisDrawing.ModelSpace.InsertBlock _  
        (insertionPnt, "CircleBlock", 1#, 1#, 1#, 0)  
    ZoomAll  
  
    ' Redefine the circle in the block,  
    ' and update the block reference  
    circleObj.radius = 3  
    blockRefObj.Update  
End Sub
```

Working with Attributes

An attribute reference provides an interactive label or tag for you to attach text to a block. Examples of data are part numbers, prices, comments, and owners' names.

You can extract attribute reference information from a drawing and use that information in a spreadsheet or database to produce items such as a parts list or bill of materials (BOM). You can associate more than one attribute reference with a block, provided that each attribute reference has a different tag. You can also define constant attributes. Because they have the same value in every occurrence of the block, AutoCAD does not prompt for a value when you insert the block.

Attributes can be invisible, which means the attribute reference is not displayed or plotted. However, information on the attribute reference is stored in the drawing file.

For more information about working with attributes, see “Overview of Block Attributes” in chapter 16, “Draw Geometric Objects,” in the *User's Guide*.

Creating Attribute Definitions and Attribute References

To create an attribute reference, first you must create an attribute definition on a block by using the `AddAttribute` method. This method requires six values as input: the height of the attribute text, the attribute mode, a prompt string, the insertion point, the tag string, and the default attribute value.

The mode value is optional. There are five constants you can enter to specify the Attribute mode:

`acAttributeModeNormal`

Specifies that the current mode of each attribute is maintained.

`acAttributeModeInvisible`

Specifies that attribute values won't appear when you insert the block. The `ATTDISP` command overrides the Invisible mode.

`acAttributeModeConstant`

Gives attributes a fixed value for block insertions.

`acAttributeModeVerify`

Prompts you to verify that the attribute value is correct when you insert the block.

`acAttributeModePreset`

Sets the attribute to its default value when you insert a block containing a present attribute. The value cannot be edited in this mode.

You can enter none, any combination, or all of the options. To specify a combination of options, add the constants together. For example, you can enter `acAttributeModeVisible + acAttributeModeConstant`.

The prompt string appears when a block containing the attribute is inserted. The default for this string is the Tag string. Input `acAttributeModeConstant` for the mode to disable the prompt.

The tag string identifies each occurrence of the attribute. You can use any characters except spaces or exclamation points. AutoCAD changes lowercase letters to uppercase.

Once the attribute definition is defined in a block, whenever you insert the block using the `InsertBlock` method you can specify a different value for the attribute reference.

An attribute definition is associated to the block upon which it is created. Attribute definitions created on model space or paper space are not considered attached to any given block.

Defining an attribute definition

This example creates a block and then adds an attribute to the block. The block is then inserted into the drawing.

```

Sub Ch10_CreatingAnAttribute()
    ' Define the block
    Dim blockObj As AcadBlock
    Dim insertionPnt(0 To 2) As Double
    insertionPnt(0) = 0
    insertionPnt(1) = 0
    insertionPnt(2) = 0
    Set blockObj = ThisDrawing.Blocks.Add _
        (insertionPnt, "BlockWithAttribute")

    ' Add an attribute to the block
    Dim attributeObj As AcadAttribute
    Dim height As Double
    Dim mode As Long
    Dim prompt As String
    Dim insertionPoint(0 To 2) As Double
    Dim tag As String
    Dim value As String
    height = 1
    mode = acAttributeModeVerify
    prompt = "New Prompt"
    insertionPoint(0) = 5
    insertionPoint(1) = 5
    insertionPoint(2) = 0
    tag = "New Tag"
    value = "New Value"
    Set attributeObj = blockObj.AddAttribute(height, mode, _
        prompt, insertionPoint, tag, value)

    ' Insert the block, creating a block reference
    ' and an attribute reference
    Dim blockRefObj As AcadBlockReference
    insertionPnt(0) = 2
    insertionPnt(1) = 2
    insertionPnt(2) = 0
    Set blockRefObj = ThisDrawing.ModelSpace.InsertBlock _
        (insertionPnt, "BlockWithAttribute", 1#, 1#, 1#, 0)
End Sub

```

Editing Attribute Definitions

You can use the Attribute object properties and methods to edit the attribute.

Some of the properties on an attribute include the following:

Alignment	Specifies the horizontal and vertical alignment of the attribute
Backward	Specifies the direction of attribute text
FieldLength	Specifies the field length of the attribute

Height	Specifies the height of the attribute
InsertionPoint	Specifies the insertion point of the attribute
Mode	Specifies the mode of the attribute
PromptString	Specifies the prompt string of the attribute
Rotation	Specifies the rotation of the attribute
ScaleFactor	Specifies the scale factor of the attribute
TagString	Specifies the tag string of the attribute

Some of the methods you can use to edit the attribute include the following:

ArrayPolar	Creates a polar array
ArrayRectangular	Creates a rectangular array
Copy	Copies the attribute
Erase	Erases the attribute
Mirror	Mirrors the attribute
Move	Moves the attribute
Rotate	Rotates the attribute
ScaleEntity	Scales the attribute

Redefining an attribute definition

This example creates a block and then adds an attribute to the block. The block is then inserted into the drawing. The attribute text is then updated to display backward.

```

Sub Ch10_RedefiningAnAttribute()
    ' Define the block
    Dim blockObj As AcadBlock
    Dim insertionPnt(0 To 2) As Double
    insertionPnt(0) = 0
    insertionPnt(1) = 0
    insertionPnt(2) = 0
    Set blockObj = ThisDrawing.Blocks.Add _
        (insertionPnt, "BlockWithAttribute")

    ' Add an attribute to the block
    Dim attributeObj As AcadAttribute
    Dim height As Double
    Dim mode As Long
    Dim prompt As String
    Dim insertionPoint(0 To 2) As Double
    Dim tag As String
    Dim value As String
    height = 1
    mode = acAttributeModeVerify
    prompt = "New Prompt"
    insertionPoint(0) = 5
    insertionPoint(1) = 5
    insertionPoint(2) = 0
    tag = "New Tag"
    value = "New Value"
    Set attributeObj = blockObj.AddAttribute(height, mode, _
        prompt, insertionPoint, tag, value)

    ' Insert the block, creating a block reference
    ' and an attribute reference
    Dim blockRefObj As AcadBlockReference
    insertionPnt(0) = 2
    insertionPnt(1) = 2
    insertionPnt(2) = 0
    Set blockRefObj = ThisDrawing.ModelSpace.InsertBlock _
        (insertionPnt, "BlockWithAttribute", 1#, 1#, 1#, 0)

    ' Redefine the attribute text to display backwards.
    attributeObj.Backward = True
    attributeObj.Update
End Sub

```

Extracting Attribute Information

You can extract attribute information from a drawing using the `GetAttributes` and `GetConstantAttributes` methods. The `GetAttributes` method returns an array of the attribute references attached to a block, along with their current values. The `GetConstantAttributes` method returns an array of constant attributes attached to the block or external reference. The attributes returned by this method are the constant attribute definitions, not attribute references.

You do not need template files to extract attribute information, and no attribute information files are created. Simply iterate the array of attribute references, using the `TagString` and `TextString` properties of the attribute reference to examine the attribute information.

The `TagString` property represents the individual tag for the attribute reference. The `TextString` property contains the value for the attribute reference.

For more information on extracting attribute information, see “Extract Block Attribute Data” in chapter 16, “Draw Geometric Objects,” in the *User’s Guide*.

Getting attribute reference information

This example creates a block and then adds an attribute to the block. The block is then inserted into the drawing. The attribute data is then returned and displayed using a message box. The attribute data is then updated for the block reference, and once again the attribute data is returned and displayed.

```

Sub Ch10_GettingAttributes()
    ' Create the block
    Dim blockObj As AcadBlock
    Dim insertionPnt(0 To 2) As Double
    insertionPnt(0) = 0
    insertionPnt(1) = 0
    insertionPnt(2) = 0
    Set blockObj = ThisDrawing.Blocks.Add _
        (insertionPnt, "TESTBLOCK")

    ' Define the attribute definition
    Dim attributeObj As AcadAttribute
    Dim height As Double
    Dim mode As Long
    Dim prompt As String
    Dim insertionPoint(0 To 2) As Double
    Dim tag As String
    Dim value As String
    height = 1#
    mode = acAttributeModeVerify
    prompt = "Attribute Prompt"
    insertionPoint(0) = 5
    insertionPoint(1) = 5
    insertionPoint(2) = 0
    tag = "Attribute Tag"
    value = "Attribute Value"

    ' Create the attribute definition object on the block
    Set attributeObj = blockObj.AddAttribute _
        (height, mode, prompt, _
        insertionPoint, tag, value)

    ' Insert the block
    Dim blockRefObj As AcadBlockReference
    insertionPnt(0) = 2
    insertionPnt(1) = 2
    insertionPnt(2) = 0
    Set blockRefObj = ThisDrawing.ModelSpace.InsertBlock _
        (insertionPnt, "TESTBLOCK", 1, 1, 1, 0)

    ZoomAll

    ' Get the attributes for the block reference
    Dim varAttributes As Variant
    varAttributes = blockRefObj.GetAttributes

    ' Move the attribute tags and values into a
    ' string to be displayed in a MsgBox
    Dim strAttributes As String
    strAttributes = ""
    Dim I As Integer
    For I = LBound(varAttributes) To UBound(varAttributes)
        strAttributes = strAttributes + " Tag: " + _
            varAttributes(I).TagString + vbCrLf + _
            " Value: " + varAttributes(I).textString
    Next

```



```

MsgBox "The attributes for blockReference " + _
        blockRefObj.Name & " are: " & vbCrLf _
        & strAttributes

' Change the value of the attribute
' Note: There is no SetAttributes. Once you have the
' variant array, you have the objects.
' Changing them changes the objects in the drawing.
varAttributes(0).textString = "NEW VALUE!"

' Get the attributes again
Dim newvarAttributes As Variant
newvarAttributes = blockRefObj.GetAttributes

' Again, display the tags and values
strAttributes = ""
For I = LBound(varAttributes) To UBound(varAttributes)
    strAttributes = strAttributes + " Tag: " + _
        newvarAttributes(I).TagString + vbCrLf + _
        " Value: " + newvarAttributes(I).textString
Next
MsgBox "The attributes for blockReference " & _
        blockRefObj.Name & " are: " & vbCrLf _
        & strAttributes
End Sub

```

Using External References

An external reference (xref) links another drawing to the current drawing. When you insert a drawing as a block, the block and all of the associated geometry is stored in the current drawing database. It is not updated if the original drawing changes. When you insert a drawing as an xref, however, the xref is updated when the original drawing changes. A drawing that contains xrefs, therefore, always reflects the most current editing of each externally referenced file.

Like a block reference, an xref is displayed in the current drawing as a single object. However, an xref does not significantly increase the file size of the current drawing and cannot be exploded. As with blocks, you can nest xrefs that are attached to your drawing.

For more information about xrefs, see “Attach, Update, and Bind External References” in chapter 23, “Reference Other Drawing Files (Xrefs),” in the *User’s Guide*.

Updating Xrefs

When you open or plot your drawing, AutoCAD reloads each xref to reflect the latest state of the referenced drawing. After you make changes to an externally referenced drawing and save the file, other users can access your changes immediately by reloading the xref.

Attaching Xrefs

Attaching an xref links one drawing (the reference file, or xref) to the current drawing. When a drawing references an xref, AutoCAD attaches only the xref definition to the drawing, unlike regular blocks, where the block definition and the contents of the block are stored with the current drawing. AutoCAD reads the reference drawing to determine what to display in the current drawing. If the reference file is missing or corrupt, its data is not displayed in the current drawing. Each time you open a drawing, AutoCAD loads all graphical and non-graphical (such as layers, linetypes, and text styles) objects from referenced files. If VISRETAIN is on, AutoCAD stores any updated xref-dependent layer information in the current drawing.

You can attach as many copies of an xref as you want, and each can have a different position, scale, and rotation. You can also control the dependent layers and linetype properties that are defined in the xref.

To attach an xref, use the `AttachExternalReference` method. This method requires you to input the path and file name of the drawing to be referenced, the name the xref is to use in the current drawing, the insertion point, the scale, and rotation information for the xref. The `AttachExternalReference` method returns the newly created `ExternalReference` object.

For more information on attaching xrefs, see “Attach External References” in chapter 23, “Reference Other Drawing Files (Xrefs),” in the *User’s Guide*.

Attaching an external reference to a drawing

This example displays all the blocks in the current drawing before and after adding an external reference. This example uses the *City map.dwg* found in the sample directory. If you do not have this image, or it is located in a different directory, insert a valid path and file name for the `PathName` variable below.

```

Sub Ch10_AttachingExternalReference()
    On Error GoTo ERRORHANDLER
    Dim InsertPoint(0 To 2) As Double
    Dim insertedBlock As AcadExternalReference
    Dim tempBlock As AcadBlock
    Dim msg As String, PathName As String

    ' Define external reference to be inserted
    InsertPoint(0) = 1
    InsertPoint(1) = 1
    InsertPoint(2) = 0
    PathName = "C:/Program Files/AutoCAD 2002/sample/City map.dwg"

    ' Display current Block information for this drawing
    GoSub ListBlocks

    ' Add the external reference to the drawing
    Set insertedBlock = ThisDrawing.ModelSpace._
        AttachExternalReference(PathName, "XREF_IMAGE", _
            InsertPoint, 1, 1, 1, 0, False)
    ZoomAll

    ' Display new Block information for this drawing
    GoSub ListBlocks
    Exit Sub
ListBlocks:
    msg = vbCrLf ' Reset message
    For Each tempBlock In ThisDrawing.Blocks
        msg = msg & tempBlock.Name & vbCrLf
    Next
    MsgBox "The current blocks in this drawing are: " & msg
    Return

ERRORHANDLER:
    MsgBox Err.Description
End Sub

```

Overlaying Xrefs

Overlaying is similar to attaching, except when a drawing is attached or overlaid. Any other overlays nested in it are ignored and, therefore, not displayed. In other words, nested overlays are not read in.

To overlay an xref, set the `bOverlay` parameter of the `AttachExternalReference` method to `TRUE`.

For more information on overlaying xrefs, see “Nest and Overlay External References” in chapter 23, “Reference Other Drawing Files (Xrefs),” in the *User’s Guide*.

Detaching Xrefs

You can detach an xref definition to remove the xrefs completely from your drawing. You can also erase the individual xref instances. Detaching the xref definition removes all dependent symbols associated with that xref. If all the instances of an xref are erased from the drawing, AutoCAD removes the xref definition the next time the drawing is opened.

To detach an xref, use the Detach method. You cannot detach a nested xref.

Detaching an xref definition

This example attaches an external reference and then detaches the external reference. This example uses the *City map.dwg* found in the sample directory. If you do not have this image, or it is located in a different directory, insert a valid path and file name for the PathName variable below.

```
Sub Ch10_DetachingExternalReference()
    On Error GoTo ERRORHANDLER

    ' Define external reference to be inserted
    Dim xrefHome As AcadBlock
    Dim xrefInserted As AcadExternalReference
    Dim insertionPnt(0 To 2) As Double
    Dim PathName As String
    insertionPnt(0) = 1
    insertionPnt(1) = 1
    insertionPnt(2) = 0
    PathName = "c:/AutoCAD/sample/City map.dwg"

    ' Add the external reference
    Set xrefInserted = ThisDrawing.ModelSpace._
        AttachExternalReference(PathName, "XREF_IMAGE", _
            insertionPnt, 1, 1, 1, 0, False)

    ZoomAll
    MsgBox "The external reference is attached."

    ' Detach the external reference definition
    Dim name As String
    name = xrefInserted.name
    ThisDrawing.Blocks.Item(name).Detach
    MsgBox "The external reference is detached."
    Exit Sub
ERRORHANDLER:
    MsgBox Err.Description
End Sub
```

Reloading Xrefs

If someone modifies an externally referenced drawing while you are working on the host drawing to which that xref is attached, you can update that xref drawing using the `Reload` method. When you reload, the selected xref drawing is updated in your host drawing. Also, if you have unloaded an xref, you can choose to reload that externally referenced drawing at any time.

Reloading an xref definition

This example attaches an external reference and then reloads the external reference. This example uses the *City map.dwg* found in the sample directory. If you do not have this image, or it is located in a different directory, insert a valid path and file name for the `PathName` variable below.

```
Sub Ch10_ReloadingExternalReference()  
    On Error GoTo ERRORHANDLER  
  
    ' Define external reference to be inserted  
    Dim xrefHome As AcadBlock  
    Dim xrefInserted As AcadExternalReference  
    Dim insertionPnt(0 To 2) As Double  
    Dim PathName As String  
    insertionPnt(0) = 1  
    insertionPnt(1) = 1  
    insertionPnt(2) = 0  
    PathName = "c:/AutoCAD/sample/City map.dwg"  
  
    ' Add the external reference to the block  
    Set xrefInserted = ThisDrawing.ModelSpace.  
        AttachExternalReference(PathName, "XREF_IMAGE", _  
            insertionPnt, 1, 1, 1, 0, False)  
    ZoomAll  
    MsgBox "The external reference is attached."  
  
    ' Reload the external reference definition  
    ThisDrawing.Blocks.Item(xrefInserted.name).Reload  
    MsgBox "The external reference is reloaded."  
    Exit Sub  
ERRORHANDLER:  
    MsgBox Err.Description  
End Sub
```

Unloading Xrefs

To unload an xref use the `Unload` method. When you unload a referenced file that is not being used in the current drawing, the AutoCAD performance is enhanced by not having to read and display unnecessary drawing geometry or symbol table information. The xref geometry and that of any nested xref is not displayed in the current drawing until the xref is reloaded.

Unloading an xref definition

This example attaches an external reference and then unloads the external reference. This example uses the *City map.dwg* found in the sample directory. If you do not have this image, or it is located in a different directory, insert a valid path and file name for the `PathName` variable below.

```
Sub Ch10_Unload ngExternal Reference()  
    On Error GoTo ERRORHANDLER  
  
    ' Define external reference to be inserted  
    Dim xrefHome As AcadBlock  
    Dim xrefInserted As AcadExternalReference  
    Dim insertionPnt(0 To 2) As Double  
    Dim PathName As String  
    insertionPnt(0) = 1  
    insertionPnt(1) = 1  
    insertionPnt(2) = 0  
    PathName = "c:/AutoCAD/sample/Ci ty map.dwg"  
  
    ' Add the external reference  
    Set xrefInserted = ThisDrawing.ModelSpace._  
        AttachExternalReference(PathName, "XREF_IMAGE", _  
            insertionPnt, 1, 1, 1, 0, False)  
  
    ZoomAll  
    MsgBox "The external reference is attached."  
  
    ' Unload the external reference definition  
    ThisDrawing.Blocks.Item(xrefInserted.name).Unload  
    MsgBox "The external reference is unloaded."  
    Exit Sub  
ERRORHANDLER:  
    MsgBox Err.Description  
End Sub
```

Binding Xrefs

Binding an xref to a drawing using the `Bind` method makes the xref a permanent part of the drawing and no longer an externally referenced file. The externally referenced information becomes a block. When the externally referenced drawing is updated, the bound xref is not updated. This process binds the entire drawing's database, including all of its dependent symbols. Dependent symbols are named objects such as blocks, dimension styles, layers, linetypes, and text styles. Binding the xref allows named objects from the xref to be used in the current drawing.

The **Bind** method requires only one parameter: `bPrefixName`. If `bPrefixName` is set to `TRUE`, the symbol names of the xref drawing are prefixed in the current drawing with `<blockname>x`, where `x` is an integer that is automatically incremented to avoid overriding existing block definitions. If the `bPrefixName` parameter is set to `FALSE`, the symbol names of the xref drawing are merged into the current drawing without the prefix. If duplicate names exist, AutoCAD uses the symbols already defined in the local drawing. If you are unsure whether your drawing contains duplicate symbol names, it is recommended that you set `bPrefixName` to `TRUE`.

For more information on binding xrefs, see “Archive Drawings That Contain External References (Bind)” in chapter 23, “Reference Other Drawing Files (Xrefs),” in the *User’s Guide*.

Binding an xref definition

This example attaches an external reference and then binds the external reference to the drawing. This example uses the *City map.dwg* found in the sample directory. If you do not have this image, or it is located in a different directory, insert a valid path and file name for the `PathName` variable below.

```
Sub Ch10_BindingExternalReference()
    On Error GoTo ERRORHANDLER

    ' Define external reference to be inserted
    Dim xrefHome As AcadBlock
    Dim xrefInserted As AcadExternalReference
    Dim insertionPnt(0 To 2) As Double
    Dim PathName As String
    insertionPnt(0) = 1
    insertionPnt(1) = 1
    insertionPnt(2) = 0
    PathName = "c:/AutoCAD/sample/City map.dwg"

    ' Add the external reference
    Set xrefInserted = ThisDrawing.ModelSpace. _
        AttachExternalReference(PathName, "XREF_IMAGE", _
            insertionPnt, 1, 1, 1, 0, False)

    ZoomAll
    MsgBox "The external reference is attached."

    ' Bind the external reference definition
    ThisDrawing.Blocks.Item(xrefInserted.name).Bind False
    MsgBox "The external reference is bound."
    Exit Sub
ERRORHANDLER:
    MsgBox Err.Description
End Sub
```

Clipping Blocks and Xrefs

There is no method provided in ActiveX Automation for clipping the boundaries of blocks and xrefs. Use the XCLIP command in AutoCAD, or send the XCLIP command to AutoCAD using the SendCommand method.

Demand Loading and Maximizing Xref Performance

Through a combination of demand loading and saving drawings with indexes, you can increase the performance of drawings with external references. Demand loading works in conjunction with the XLOADCTL and INDEXCTL system variables. When you turn on demand loading, if indexes have been saved in the referenced drawings, AutoCAD loads into memory only the data from the reference drawing that is necessary to regenerate the current drawing. In other words, referenced material is read in “on demand.”

To realize the maximum benefits of demand loading, you need to save the referenced drawings with layer and spatial indexes. The performance benefits of demand loading are most noticeable when you

- Clip the xref to display a small fraction of it, and a spatial index is saved in the externally referenced drawing.
- Freeze several layers of the xref, and the externally referenced drawing is saved with a layer index.

To turn on demand loading, use the XRefDemandLoad property. If you turn on demand loading with the `acDemandLoadEnabledWithCopy` option, AutoCAD makes a temporary copy of the externally referenced file and demand loads the temporary file. You can then demand load the xref while allowing the original reference drawing to be available for modification. When you disable demand loading, AutoCAD reads in the entire reference drawing regardless of layer visibility or clip instances.

To turn on layer and spatial indexes, set the INDEXCTL system variable using the SetVariable method. The following settings apply to the INDEXCTL system variable:

- 0 = no indexes created
- 1 = layer index created
- 2 = spatial index created
- 3 = both spatial and layer indexes created

By default, INDEXCTL is set to 0 when you create a new AutoCAD drawing.

For more information on demand loading and xrefs, see “Increase Performance with Large Xrefs” in chapter 23, “Reference Other Drawing Files (Xrefs),” in the *User’s Guide*.

Assigning and Retrieving Extended Data

You can use extended data (xdata) as a means for linking information with objects in a drawing.

Assigning xdata to all objects in a selection set

This example prompts the user to select objects from the drawing. The selected objects are placed into a selection set, and the specified xdata is attached to all objects in that selection set.

```
Sub Ch10_AttachXDataToSelectionSetObjects()  
    ' Create the selection set  
    Dim sset As Object  
    Set sset = ThisDrawing.SelectionSets.Add("SS1")  
  
    ' Prompt the user to select objects  
    sset.SelectOnScreen  
  
    ' Define the xdata  
    Dim appName As String, xdataStr As String  
    appName = "MY_APP"  
    xdataStr = "This is some xdata"  
    Dim xdataType(0 To 1) As Integer  
    Dim xdata(0 To 1) As Variant  
  
    ' Define the values for each array  
    ' 1001 indicates the appName  
    xdataType(0) = 1001  
    xdata(0) = appName  
    ' 1000 indicates a string value  
    xdataType(1) = 1000  
    xdata(1) = xdataStr  
  
    ' Loop through all entities in the selection  
    ' set and assign the xdata to each entity  
    Dim ent As Object  
    For Each ent In sset  
        ent.SetXData xdataType, xdata  
    Next ent  
End Sub
```

Viewing the xdata of all objects in a selection set

This example displays the xdata attached with the previous example. If you attach xdata other than strings (type 1000), you will need to revise this code.

```
Sub Ch10_ViewXData()  
    ' Find the selection created in previous example  
    Dim sset As Object  
    Set sset = ThisDrawing.SelectionSets.Item("SS1")  
  
    ' Define the xdata variables to hold xdata information  
    Dim xdataType As Variant  
    Dim xdata As Variant  
    Dim xd As Variant  
  
    ' Define index counter  
    Dim xdi As Integer  
    xdi = 0  
  
    ' Loop through the objects in the selection set  
    ' and retrieve the xdata for the object  
    Dim msgstr As String  
    Dim appName As String  
    Dim ent As AcadEntity  
    appName = "MY_APP"  
    For Each ent In sset  
        msgstr = ""  
        xdi = 0  
  
        ' Retrieve the appName xdata type and value  
        ent.GetXData appName, xdataType, xdata  
  
        ' If the xdataType variable is not initialized, there  
        ' was no appName xdata to retrieve for that entity  
        If VarType(xdataType) <> vbEmpty Then  
            For Each xd In xdata  
                msgstr = msgstr & vbCrLf & xdataType(xdi) _  
                    & ": " & xd  
                xdi = xdi + 1  
            Next xd  
        End If  
  
        ' If the msgstr variable is NULL, there was no xdata  
        If msgstr = "" Then msgstr = vbCrLf & "NONE"  
        MsgBox appName & " xdata on " & ent.ObjectName & _  
            ": " & vbCrLf & msgstr  
    Next ent  
End Sub
```

Developing Applications with VBA

11

Many programming tasks involve more than simply working with the AutoCAD ActiveX object model. This chapter provides a brief overview of creating dialog boxes, handling errors, controlling window focus, and distributing your application to others.

Remember, the Microsoft documentation for VBA contains more information on these topics.

In this chapter

- More VBA Terminology
- Working with Forms in VBA
- Handling Errors
- Encrypting VBA Code Modules
- Running a VBA Macro from a Toolbar or Menu
- Automatically Loading a VBA Project
- Automatically Running a VBA Macro
- Automatically Opening the VBA IDE Whenever a Project Is Loaded
- Working in a Zero Document State
- Distributing Your Application

More VBA Terminology

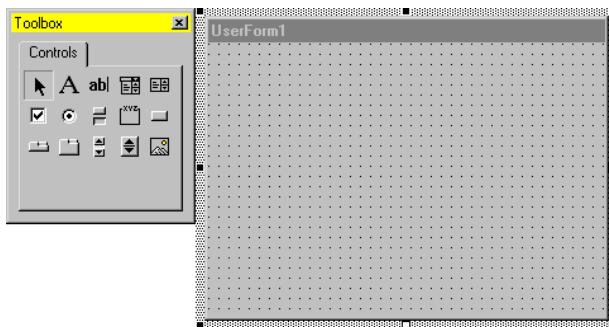
This chapter expands your exposure to VBA. The following terms will help you understand and work within the VBA environment.

Project	A set of forms and modules grouped together in a single file.
Module	A group of (usually related) subroutines and functions.
Macro	A public subroutine or function. Macros are exposed to the user as an executable component of your project.
Dialog box	A means by which information is displayed or gathered during application execution.
Form	Container for dialog box controls.

Working with Forms in VBA

Forms are the basic building blocks through which you create your own custom dialog boxes for your application. Through custom forms you can provide information to users, get information from users, or have your users control activity in the application.

Forms are like an artist's canvas—they start out blank. To fill your canvas, you need a palette. In this case, your palette is the control toolbox. You, as the artist, place selected controls from the toolbox onto the form. You can add as many controls as you like. At any time you can adjust size and properties of both the controls and even the form itself. Finally, you add the functionality (code) to the controls that brings your form to life.



Although Visual Basic supports different types of forms, VBA supports only the UserForm. This means some forms have been created and exported in Visual Basic that cannot be imported into VBA.

UserForms—or forms, as they are called in this guide—can be modal or modeless. The ShowModal property of a form determines whether it is modal or modeless. Modal forms displayed in your running application must be closed before users can perform any other action in the application. For more information about working with modal forms, see “Designing Your Application for Use with Modal Forms” on page 317.

To create a new form in your project

- 1 Open the Project window of the VBA IDE and select the project you want to add the form to.
- 2 From the Insert menu, choose UserForm.

A blank form is created and added to your project.

Designing in Design Mode, Running in Run Mode

While you are building your form you are working in design mode, where you can

- Add controls to the form
- Change the properties of the form
- Change the properties of controls on the form
- Add code to the form module

While in design mode there is no interaction among the user, the user interface of AutoCAD, and your form.

Once you run your application, or your user runs your application, the form is then in run mode. While in run mode you cannot make adjustments to the form directly. However, the form is now displayed in the AutoCAD user interface and the user can interact with the form as part of the normal operation of your application.

Adding Controls to a Form

Adding controls to a form is easy. Simply select a control from the control toolbox and drag it over to the form. When you release your mouse, a copy of the control will be placed on the form. Once the control is on the form, you can change the position and size of the control. You can copy over as many controls as you like.

In addition to the drag method previously mentioned, there are other ways of placing controls on a form.

Changing the Size and Placement of a Control

To move a control, simply select it and drag it to its new position on the form.

To resize a control, first select the control by clicking on it once. When you select a control its border becomes visible. To resize the control, simply select one of the sizing grips now visible on the border and drag the grip to a new position. When you release the grip, the control will resize to that location. (You can resize the form in the same manner.)

To move or resize several controls at once, select each control while holding down the SHIFT key. This will highlight all the controls. You can now move or resize the controls as a group.

To size a control as you place it

- 1 Select the desired control in the control toolbox.
- 2 On the form, press, drag, and release the mouse button. The selected control will be placed on the form. The size of the control depends on how far you drag the mouse.

To place several instances of the same control

- 1 In the control toolbox, double-click on the control you want to place.
- 2 On the form, click at the location you want a copy of the control placed. Move to another location on the form and click again. Another copy of the control will appear. You can add as many copies of the control as you need.
- 3 When you are finished with the control, return to the control toolbox and click the control one more time to deselect it.

Using Formatting Controls

VBA provides several formatting controls to help you lay out your form. These controls can be found on the Format menu of the VBA IDE. These controls allow you to align controls to each other, make two or more controls the same size, change the spacing between controls, and center controls on the form.

Remember when using the formatting controls that several controls can be selected at once by using **SHIFT**.

Changing the Properties of a Control

Properties control various characteristics of a control such as its size, shape, color, label, and default values. You can set the properties of a control in design mode by using the Properties window.

To change the property of a control

- 1 On the form, select the desired control.
- 2 Open the Properties window using **F4** if it is not already open.
- 3 In the Properties window, find the property you want to change and select the current value for that property.
- 4 Change the value to the new desired value for the property.

You can also change the property of a control at runtime by writing code to access that property. See the Microsoft documentation for more information on changing the property of a control at runtime.

Adding Code to a Control

Now that you have your form looking the way you want, it's time to add some code behind your controls. To open the Code window for a control simply double-click on the control in the Form window. The Code window will open, with a subroutine created for that control and its default event.

You can add code to the default event, or choose a different event from the event drop-down list at the top-right corner of the Code window.

Displaying and Hiding Forms

Now you have a beautifully designed form with fully functional code behind all the controls. The last step is getting the form displayed to the user at runtime. Displaying the form is accomplished through the VBA **Show** method. The **Show** method can be called from any code module in your application.

The form you created is modal by default, so the user will not be able to interact with AutoCAD directly while the form is displayed. For example, the user cannot select a point or object in the drawing with the form displayed. To allow the user access to the AutoCAD drawing, use the VBA Hide method. The Hide method hides the form and allows the user limited access to AutoCAD. When using the Hide method it is important to remember the form is not unloaded from memory. It will retain all current values while hidden.

The Hide method is called in the same manner as the Show method.

Displaying a form

This example will display the form named “UserForm1”:

```
Public Sub MyApplication()  
    UserForm1.Show  
End Sub
```

The subroutine (and consequently the display of your form) is now callable as a macro from the VBARUN command or from an AutoCAD menu.

Hiding a form

This example hides the form named “UserForm1”:

```
Public Sub MyAppHide()  
    UserForm1.Hide  
End Sub
```

Loading and Unloading Forms

There may be times when you want to load a form into memory during run-time, but not show the form. You may choose to do this to better control when the load time occurs in your application, or when you need programmatic access to the form but do not want to display the form to the user.

To load a form, but not display it, use the VBA Load method. The Show method can then be used to make the form visible at the appropriate time in your application’s execution. Remember, the user can’t interact with your form until it is visible.

If the Show method is called and the form has not been loaded, it will be loaded automatically.

There may also be times when you will want to unload a form specifically. Unloading a form removes that form from memory and all the memory associated with the form is reclaimed. Until the form is loaded again by using either the Load or Show method, a user can’t interact with the form, and the

form can't be manipulated programmatically. You may choose to unload a form when you know the form will not be used again in the application and you want to reclaim the memory.

The Hide method does not perform an unload. If your application ends and a form has not been unloaded, it will be unloaded automatically. The following table compares the VBA Show, Hide, Load, and Unload methods:

VBA Show, Hide, Load, and Unload methods	
Method	Use
Show	Displays a form. If the form has not been loaded, it is loaded automatically.
Hide	Hides a form. The form is not unloaded from memory.
Load	Loads a form into memory but does not display it.
Unload	Unloads a form from memory. This can be done explicitly from the Unload method, or automatically at the termination of the application.

Designing Your Application for Use with Modal Forms

When you define a dialog box as modal in AutoCAD VBA, the user must respond to the dialog box before any other part of the application is allowed to continue. No subsequent code is executed until the modal dialog box is closed through either the Hide or Unload method. This requires that you, as the developer of the application, think carefully about how and when you implement dialog boxes.

For example, you may have a dialog box that requires the user to select an object in the AutoCAD drawing. For the user to be able to pick the object from the AutoCAD Application window you must hide the form by calling the Hide method. Once the object has been selected you use the Show method to redisplay the form, with all of its data still current, and continue with the application.

NOTE Although other forms in the application are disabled when a modal dialog box is displayed, other applications are not.

Handling Errors

Most development environments provide default error handling. For VB and VBA, the default reaction to an error is to display an error message and terminate the application. While this behavior is adequate during the development phase of your application, it is not productive for your end-user. There may be errors that you want to ignore, or that you want to provide special responses to. There may be errors that you will want to suppress the error message display for, or simply control the message that gets displayed to the user. In addition, automatically terminating the application is hardly ever acceptable to the end-user.

In general, error handling is necessary whenever user-input is required and whenever working with file I/O. Remember, even if you are sure a needed file is there and available for processing, there may be conditions you haven't thought of that could cause errors.

NOTE Most of the code examples provided in the AutoCAD documentation do not use error trapping. This keeps the examples simple and to the point. However, as with all programming languages, proper error trapping and handling is essential for a robust application.

Defining Application Error Types

There are three different types of errors you can encounter in your applications: compile-time errors, runtime errors, and logic errors.

- **Compile-time errors** occur during the construction of your application. These errors consist mostly of syntax mistakes, variable scoping problems, or data typing problems. In VBA, these types of errors are caught by the development environment. When you enter an incorrect line of code, the line is highlighted and an error message appears telling you the problem. Compile-time errors must be corrected before the application can run.
- **Runtime errors** are a little more difficult to find and correct. They occur during the execution of your code, and often involve receiving information from the user. For example, if your application requires the user to enter the name of a drawing and the user enters a name for a drawing that didn't exist, a runtime error occurs. To handle runtime errors effectively, you must predict what kinds of problems could happen, trap them, and then write code to handle these situations.

- Logic errors are the most difficult to find and correct. Symptoms of logic errors include situations in which there are no compile-time errors and no runtime errors, but the outcome of your program is still incorrect. This is what programmers refer to as a bug—and a bug can be very easy or very difficult to track down.

Information on finding and correcting all three types of errors can be found in documentation for your development environment. AutoCAD-specific errors fall into the runtime error category, so these types of errors will be covered more fully in this documentation.

Trapping Runtime Errors

In VB and VBA, runtime errors are trapped using the `On Error` statement. This statement literally sets a trap for the system. When an error occurs, this statement automatically detours processing to your specially written error handler. The default error handling for the system is bypassed.

The `On Error` statement has three forms:

- `On Error Resume Next`
- `On Error GoTo Label`
- `On Error GoTo 0`

The `On Error Resume Next` statement is used when you want to ignore errors. This statement traps the error and instead of displaying an error message and terminating the program, it simply moves on to the next line of code and continues processing. For example, if you wanted to create a sub-routine to iterate through model space and change the color of each entity, you know that AutoCAD will throw an error if you try to color an entity on a locked layer. Instead of terminating the program, simply skip the entity on the locked layer and continue processing the remaining entities. The `On Error Resume Next` statement lets you do just that.

The `On Error GoTo Label` statement is used when you want to write an explicit error handler. This statement traps the error and instead of displaying an error message and terminating the program, it jumps to a specific location in your code. Your code can then respond to the error in whatever manner is appropriate for your application. For example, you can expand the example above to display a message containing the handle for each entity on the locked layer.

Handling errors with the On Error Resume Next statement

The following subroutine iterates model space and changes the color of each entity to red. Try running this subroutine on a drawing with several entities, some of which are on a locked layer. Next, comment out the `On Error Resume Next` statement and run the subroutine again. You will notice the subroutine terminates at the first entity on the locked layer.

```
Sub Ch11_ColorEntities()  
    Dim entry As Object  
    On Error Resume Next  
    For Each entry In ThisDrawing.ModelSpace  
        entry.Color = acRed  
    Next entry  
End Sub
```

Handling errors with the On Error GoTo statement

The following subroutine iterates model space and changes the color of each entity to red. For each entity on the locked layer, the error handler displays a custom error message and the handle of the entity. Try running this subroutine on a drawing with several entities, some of which are on a locked layer. Next, comment out the `On Error GoTo MyErrorHandler` statement and run the subroutine again. You will notice the subroutine terminates at the first entity on the locked layer.

```
Sub Ch11_ColorEntities2()  
    Dim entry As Object  
    On Error GoTo MyErrorHandler  
    For Each entry In ThisDrawing.ModelSpace  
        entry.Color = acRed  
    Next entry  
    ' Important! Exit the subroutine before the error handler  
    Exit Sub  
MyErrorHandler:  
    MsgBox entry.EntityName + " is on a locked layer." + _  
        " The handle is: " + entry.Handle  
    Resume Next  
End Sub
```

The `On Error GoTo 0` statement cancels the current error handler. The `On Error Resume Next` and `On Error GoTo Label` statements remain in effect until the subroutine ends, another error handler is declared, or the error handler is canceled with the `On Error GoTo 0` statement.

Responding to Trapped Errors

Now that you have trapped an error, what do you do with it? The answer depends on the nature of your application and the nature of the error.

VB and VBA provide information on the type of error that has been trapped by using the Err object. This object has several properties: Number, Description, Source, HelpFile, HelpContext, and LastDLLError. The properties of the Err object get filled in with the information for the most current error. The most important properties are the Number and Description properties. The Number property contains the unique error code associated with the error, and the Description property contains the error message that would normally be displayed.

In your error handler you can compare the Number property of the error to an expected value. This will help you determine the nature of the error that has occurred. Once you know what kind of error you are dealing with, you can take the appropriate action.

Responding to AutoCAD User Input Errors

The user-input methods provide a certain amount of inherent error trapping in that they require the user to enter a certain type of data. If the user tries to enter some other data, AutoCAD rejects the input and reprompts the user. Using the InitializeUserInput method with the user input functions provides additional control of the user input but can also introduce additional conditions that must be verified through error trapping. For an example of error trapping that is required with certain types of user-input, see “Prompting for User Input” on page 75.

Encrypting VBA Code Modules

Although VBA does not support creation of executables, it does offer password protection for the visibility of the project forms, classes, and modules on a project basis. You can find this Project Protection facility in the VBA IDE menu. Choose Tools ► Project Properties ► Protection.

Running a VBA Macro from a Toolbar or Menu

You can run a VBA macro from an AutoCAD toolbar or menu by simply changing the Macro property for that toolbar or menu. The Macro property must be set equal to

```
-VBARUN filename.dvb!modulename.macroname
```

where *filename* is the name of the project file, *modulename* is the name of the module containing the macro to be run, and *macroname* is the name of the macro. The file name is only required when the file is not loaded in the current session of AutoCAD. If the file name is provided, the file will be loaded.

For more information on editing menus and toolbars, see chapter 6, “Customizing Toolbars and Menus.”

Automatically Loading a VBA Project

There are two different ways to load a VBA project automatically:

- When VBA is loaded it will look in the AutoCAD directory for a project named *acad.dvb*. This file will automatically load as the default project
- Any project other than the default, *acad.dvb*, can be used by explicitly loading that project at startup using the VBALOAD command. The following code sample uses the AutoLISP startup file to load VBA and a VBA project named *myproj.dvb* when AutoCAD is started. Start *notepad.exe* and create (or append to) *acad.lsp* the following lines:

```
(defun S: : STARTUP()  
  (command "_VBALOAD" "myproj.dvb")  
)
```

Automatically Running a VBA Macro

You can automatically run any macro in the *acad.dvb* file by calling it with the command line version of VBARUN from an AutoCAD startup facility like *acad.lsp*. For example, to automatically run the macro named *drawline*, first save the *drawline* macro in the *acad.dvb* file. Next, invoke *notepad.exe* and create (or append to) *acad.lsp* the following lines:

```
(defun S: : STARTUP()  
  (command "_-vbarun" "drawline")  
)
```

You can cause a macro to run automatically when VBA loads by naming the macro AcadStartup. Any macro in your *acad.dvb* file called AcadStartup will automatically get executed when VBA loads.

Automatically Opening the VBA IDE Whenever a Project Is Loaded

There is an option on the Open VBA Project dialog box that allows you to open the IDE automatically. Simply check the box “Open Visual Basic Editor” found in the lower-left side of the dialog box and the VBA IDE will open automatically whenever a VBA project is loaded. This option will remain set until you turn it off again.

NOTE To access the Open VBA Project dialog box, enter VBALOAD at the command line. The dialog box will open and allow you to choose a project to load. If you do not see the Open VBA Project dialog box, it is most likely because the system variable FILEDIA is turned off. This system variable turns on and off the display of dialog boxes. To turn FILEDIA back on, set it to 1.

Working in a Zero Document State

A zero document state is when there are no open drawings in AutoCAD. There are several important considerations to keep in mind when you are working with VBA in a zero document state:

- The `Thi sDrawi ng` object is undefined in a zero document state. Any attempt to use `Thi sDrawi ng` will result in an error.
- Objects that are document dependent are also not defined in a zero document state. Document dependent objects are those objects that fall below the Document object in the AutoCAD object model. Working with non-document dependent objects, such as the Application or MenuBar objects, is allowed.
- AutoCAD does not have a command line in a zero document state. Any attempt to access the AutoCAD command line while AutoCAD is in a zero document state will result in an error.

Distributing Your Application

VBA applications can be distributed two different ways:

- Embedded in an AutoCAD drawing
- Stored in a VBA project file

You must choose a distribution option that is appropriate for your application. Applications that are applicable to the current drawing, and do not access other drawings, are often embedded in the drawing. By embedding the application in the drawing, you can always be sure the application is loaded, and therefore available to the user whenever the drawing is open.

Applications that are used by many people, are updated frequently, need to open and close other drawings, or are not used frequently you may want to store in a VBA project file. In this way, there is one central location for the application, and everyone can be sure to use the latest version.

For more information on embedded projects and VBA project files, see “Understanding Embedded and Global VBA Projects” on page 12.

Distributing Visual Basic Applications

Visual Basic applications, or any other out-of-process application, cannot be stored within an AutoCAD drawing. These applications are compiled into standalone executables (EXE) and can be run from AutoCAD using the `APPLOAD` command.

Interacting with Other Applications and Windows APIs

ActiveX technology allows you to exchange information easily with other AutoCAD applications and other ActiveX-enabled applications such as Microsoft Excel or Microsoft Word. This chapter examines some of the basic procedures for interacting with other applications.

12

In this chapter

- Interacting with Visual LISP Applications
- Interacting with Other Windows Applications
- Accessing Windows APIs from VBA

Interacting with Visual LISP Applications

Visual LISP applications have access to the entire range of ActiveX objects. They can call ActiveX methods, and set and retrieve ActiveX properties. In addition, Visual LISP applications can also run VBA macros through the VBARUN command.

ActiveX and VBA applications can execute Visual LISP applications through the SendCommand method. This method allows ActiveX and VBA applications to send a command to the AutoCAD command line.

For more information about accessing ActiveX objects through Visual LISP, see the *Visual LISP Developer's Guide*.

Interacting with Other Windows Applications

AutoCAD ActiveX technology allows you to exchange information easily with other ActiveX-enabled applications such as Microsoft Excel or Microsoft Word. This capability allows you to collect, store, and present AutoCAD information in formats other than the AutoCAD drawing. You can also read information from these applications back into AutoCAD to direct the creation or manipulation of AutoCAD objects. An example of using this technology is to create a bill of materials as a Microsoft Excel spreadsheet from the objects in an AutoCAD drawing.

You have already learned how to write code using the AutoCAD ActiveX Object Model. Exchanging information with other ActiveX-enabled applications involves simply referencing the other applications' ActiveX Object Model and writing the code necessary to utilize their objects.

NOTE This chapter provides only a brief introduction to the capabilities of cross-application programming. This material is not AutoCAD specific, and as such it is discussed in both Microsoft documentation and independent programming guides.

To exchange information across ActiveX Object Models

- 1 Reference the other applications' ActiveX Object Model.
This will make your code aware of the names and relationships of the objects in the other Object Model.
- 2 Create an instance of the other application.
This will create valid objects for (instantiate) the basic objects in the other Object Model.
- 3 Write your code utilizing both the AutoCAD Object Model and the other applications' Object Model.
This is where the exchange of data takes place.

Referencing the ActiveX Object Library of Other Applications

To write code that accesses another application, you must instruct VBA to make the objects in the other application available to you. You do this by setting a reference in the other application's object library. This is a file on your computer where all the objects, methods, properties, constants, and events for that application are defined.

You make a reference to an object library through the VBA IDE. In the VBA IDE, under the Tools menu, there is a menu option called References. This menu option will bring up a dialog box that lists all of the object libraries VBA finds on your system. To make a reference to a library, simply select the library from the list. Libraries with check boxes that are selected are already referenced in the current project. For example, to add the Microsoft Excel object library, select the Microsoft Excel object library entry in the list.

Once you have created a reference to another application's object library, you can use the VBA Object Browser to view a list of the application's objects.

NOTE You must set the reference for each VBA project that will use this Object Model. Setting the reference for one project won't automatically set it for another project. This is for performance reasons.

To make a reference to another application's object library

- 1 In the VBA IDE, open the Tools menu and select the References menu option.
- 2 Find and select the entry in the list of Available References for the application you want to access.
- 3 Select OK to close the dialog box with your changes.

Creating an Instance of the Other Application

Once you have referenced an application's object library you must create an instance of the application. This is just a fancy way of saying you need to start the other application programmatically so your code will have valid objects to work with.

To do this, first declare a variable that will represent the other application. You do this the same way as built-in objects, by using a `Dim` statement. You should qualify the type of application in your `Dim` statement. For example, this `Dim` statement declares an object variable of type `Excel . Application`:

```
Dim ExcelAppObj as Excel . Application
```

After you declare the variable, use the `Set` statement with the `New` keyword to set the variable equal to a running instance of the application. For example, the following `Set` statement sets the variable declared above equal to the Excel application. The `New` keyword starts a new session of Excel.

```
Set ExcelAppObj = New Excel . Application
```

NOTE Some applications allow only one running instance of the application at a time. Using the `New` keyword on such an application will establish a reference to the existing instance and will not launch a new session of the application.

Programming with Objects from Other Applications

Now that you have referenced the object library and created a new instance of the application, you can create and manipulate objects in that application. All the objects, methods, and properties defined by the Object Model are available to you. For example, using the variable declarations from the previous section, the following line of code makes the Excel session visible to the user:

```
ExcelAppObj.Visible = TRUE
```

You should familiarize yourself with the Object Model of the application you are writing code for. You can use the VBA Object Browser or the application's help file to learn about any Object Model you are referencing.

Quitting the Other Application

When you start an application programmatically it takes up memory in the computer. You should quit the application when you have finished using it so system resources can be freed up.

Although each Object Model is different, most have a Quit method from the Application object that can be used to close the application cleanly. For example, using the variable declarations from the previous section, the following line of code will quit Excel:

```
ExcelAppObj.Application.Quit
```

NOTE Destroying or going beyond the scope of the object variable does not necessarily cause the application to terminate. You should always quit the application using the appropriate method to assure proper memory cleanup.

Listing AutoCAD attributes on an Excel spreadsheet

This subroutine finds all the block references in the current drawing. It then finds the attributes attached to those block references and lists them in an Excel spreadsheet. To run this example, do the following:

- 1 Open a drawing containing block references with attributes. (The sample drawing *sample/activeX/attrib.dwg* contains such block references.)
- 2 Open the VBA IDE using the AutoCAD VBAIDE command.
- 3 Using the Tools ► References menu option in the VBA IDE, select Microsoft Excel 8.0 Object Model.
- 4 Copy this subroutine into a VBA Code window and run it.

```

Sub Ch12_Extract()
    Dim Excel As Excel.Application
    Dim ExcelSheet As Object
    Dim ExcelWorkbook As Object

    Dim RowNum As Integer
    Dim Header As Boolean
    Dim elem As AcadEntity
    Dim Array1 As Variant
    Dim Count As Integer

    ' Launch Excel.
    Set Excel = New Excel.Application

    ' Create a new workbook and find the active sheet.
    Set ExcelWorkbook = Excel.Workbooks.Add
    Set ExcelSheet = Excel.ActiveSheet
    ExcelWorkbook.SaveAs "Attribute.xls"

    RowNum = 1
    Header = False
    ' Iterate through model space finding
    ' all block references.
    For Each elem In ThisDrawing.ModelSpace
        With elem
            ' When a block reference has been found,
            ' check it for attributes
            If StrComp(.EntityName, "AcDbBlockReference", 1) _
                = 0 Then
                If .HasAttributes Then
                    ' Get the attributes
                    Array1 = .GetAttributes
                    ' Copy the Tagstrings for the
                    ' Attributes into Excel
                    For Count = LBound(Array1) To UBound(Array1)
                        If Header = False Then
                            If StrComp(Array1(Count).EntityName, _
                                "AcDbAttribute", 1) = 0 Then
                                ExcelSheet.Cells(RowNum, _
                                    Count + 1).value = _
                                    Array1(Count).TagString
                            End If
                        End If
                    Next Count
                    RowNum = RowNum + 1
                    For Count = LBound(Array1) To UBound(Array1)
                        ExcelSheet.Cells(RowNum, Count + 1).value = _
                            Array1(Count).textString
                    Next Count
                    Header = True
                End If
            End If
        End With
    Next elem
    Excel.Application.Quit
End Sub

```

Accessing Windows APIs from VBA

The Windows API procedures are available to most Windows applications. These procedures allow you to expand the capabilities of your application.

Through the Windows APIs you can obtain information about the current system, such as which other programs are installed or running on the system, where information is located on a system, and what the current control settings are for the system. You can also access joystick, multimedia, and sound controls. These tasks represent but a few of the many capabilities provided by the Windows APIs.

To use a Windows API, you must first declare the API in your application. This is done through the Visual Basic `Declare` statement. The `Declare` statement requires several pieces of information:

- The name of the dynamic link library (DLL) containing the procedure you want to use
- The name of the procedure as it appears in the DLL
- The name of the procedure as you want to use it in your application
- The parameters the procedure expects to receive
- The return value data type (if the procedure you are calling is a function)

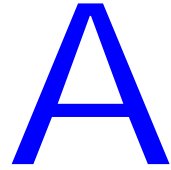
You can place the `Declare` statement in any of your VBA modules. If you place it in a standard module, the procedure will be available to any module in your application, unless you limit its scope by using the keyword `Private`. If you place the `Declare` statement in a class or form module, the procedure will only be available in that module. Once a procedure has been declared, you can call that procedure as you would any other procedure in your application.

Getting a `Declare` statement just right is a difficult skill to learn. Getting a `Declare` statement wrong is easy, but it often comes with dire consequences. Be sure to save any information in active applications before you try out a new `Declare` statement.

To help you with your `Declare` statements, Microsoft provides a file listing of many of the declarations most commonly used. The file is called *Win32api.txt* and comes with Visual Basic and Office. You can search this file for the procedure you need and copy the `Declare` statement provided into your code.

The Microsoft VBA documentation contains more information on the Declare statement and an example of its use. The Microsoft Windows API Reference is available as part of the Microsoft Developer Network CD subscription and provides a reference to all the available procedures in the Windows APIs. Dan Appleman's book *Visual Basic Programmer's Guide to the Win32 API* is also an excellent resource directed at the Visual Basic programmer.

Visual LISP and ActiveX/ VBA Comparison



Most of the capabilities found in the Visual LISP interface can also be found in the ActiveX and VBA interface. This appendix serves as a reference to help developers familiar with Visual LISP find the equivalent ActiveX and VBA functionality.

In this appendix

- AutoLISP and ActiveX/VBA Comparison

AutoLISP and ActiveX/VBA Comparison

The following table compares AutoLISP functions with the similar ActiveX or Visual Basic functions and operators. The ActiveX Automation equivalents are indicated by “AutoCAD.Application.” and the Visual Basic equivalents are listed as a function or operator.

Paper space, model space, and TILEMODE settings	
AutoLISP function	ActiveX or Visual Basic equivalent
+ (addition)	+ (addition operator)
– (subtraction)	– (subtraction operator)
* (multiplication)	* (multiplication operator)
/ (division)	/ (division operator)
= (equal to)	= (equal to comparison operator)
/= (not equal to)	<> (not equal to comparison operator)
< (less than)	< (less than comparison operator)
<= (less than or equal to)	<= (less than or equal to comparison operator)
/= (not equal to)	<> (not equal to comparison operator)
> (greater than)	> (greater than comparison operator)
>= (greater than or equal to)	>= (greater than or equal to comparison operator)
~ (bitwise not)	Not operator
1+ (increment)	Use + (addition operator)
1– (decrement)	Use – (subtraction operator)
abs	Abs function
acad_colordlg	<i>Not provided</i>
acad_helpdlg	Search for HELP in the online Help index
acad_strlsort	Search for SORT in the online Help index

Paper space, model space, and TILEMODE settings (continued)

AutoLISP function	ActiveX or Visual Basic equivalent
action_tile	Use the Visual Basic Dialog Editor
add_list	Use the Visual Basic Dialog Editor
ads	AutoCAD.Application.ListADS method
alert	MsgBox function
and	And operator
angle	AutoCAD.Application.ActiveDocument.Utility. AngleFromXAxis method
angtof	AutoCAD.Application.ActiveDocument.Utility.AngleToReal method
angtos	AutoCAD.Application.ActiveDocument.Utility. AngleToString method
append	Use Visual Basic array manipulation functions
apply	<i>Not provided</i>
arx	AutoCAD.Application.ListARX method
arxload	AutoCAD.Application.LoadARX method
arxunload	AutoCAD.Application.UnloadARX method
ascii	Asc function
assoc	<i>Not provided</i>
atan	Atn function
atof	CDbl Function
atoi	CInt Function
atom	Search for IS in the online Help index
atoms-family	<i>Not provided</i>
autoarxload	<i>Not provided</i>
autoload	<i>Not provided</i>

Paper space, model space, and TILEMODE settings (continued)

AutoLISP function	ActiveX or Visual Basic equivalent
Boole	Use Visual Basic logical operators
boundp	Search for IS in the online Help index
car/cdr	Use Visual Basic array manipulation functions
chr	Chr function
client_data_tile	Use the Visual Basic Dialog Editor
close	AutoCAD.Application.Documents.Close method
command	AutoCAD.ActiveDocument.SendCommand method
cond	Select Case statement
cons	Use array manipulation functions or AutoCAD.Application.collection.Add<entityname> method
cos	Cos function
cvunit	Use the conversion functions
defun	The Visual Basic keywords Function and End Function
dictadd	AutoCAD.Application.ActiveDocument.Dictionaries.Add method
dictnext	AutoCAD.Application.ActiveDocument.Dictionaries.Item method
dictremove	AutoCAD.Application.ActiveDocument.Dictionaries. Dictionary.Delete method
dictrename	AutoCAD.Application.ActiveDocument.Dictionaries. Dictionary.Rename method
dictsearch	AutoCAD.Application.ActiveDocument.Dictionaries. Dictionary.GetName and GetObject methods
dimx_tile and dimy_tile	Use the Visual Basic Dialog Editor
distance	AutoCAD.Application.Utility.GetDistance for interactive method.
distof	<i>Not provided</i>

Paper space, model space, and TILEMODE settings (continued)

AutoLISP function	ActiveX or Visual Basic equivalent
done_dialog	Use the Visual Basic Dialog Editor
end_image	Use the Visual Basic Dialog Editor
end_list	Use the Visual Basic Dialog Editor
entdel	AutoCAD.Application.ActiveDocument.collection_object. Delete method
entget	AutoCAD.Application.ActiveDocument.collection_object. property properties
entlast	AutoCAD.Application.ActiveDocument.Modelspace. Item(count-1)
entmake	AutoCAD.Application.ActiveDocument.Modelspace. Add<entityname> method
entmakex	AutoCAD.Application.ActiveDocument.Modelspace. Add<entityname> method
entmod	Use any of the read-write properties for the object
entnext	AutoCAD.Application.ActiveDocument.collection.Item method
entsel	AutoCAD.Application.ActiveDocument.SelectionSets object/methods/properties
entupd	AutoCAD.Application.ActiveDocument.Modelspace.object. Update method
eq	<i>Not provided</i>
equal	Eqv operator
error	Error object/method/properties
eval	<i>Not provided</i>
exit	AutoCAD.Application.Quit method
exp	Exp function
expand	<i>Not provided</i>

Paper space, model space, and TILEMODE settings (continued)

AutoLISP function	ActiveX or Visual Basic equivalent
expt	^ (exponentiation operator)
fill_image	Use the Visual Basic Dialog Editor
findfile	Dir function
fix	Fix, Int, CInt functions
float	CDBl Function
foreach	For Each...Next statement
gc	AutoCAD.Application.ActiveDocument.PurgeAll
gcd	<i>Not provided</i>
get_attr	Use the Visual Basic Dialog Editor
get_tile	Use the Visual Basic Dialog Editor
getangle	AutoCAD.Application.ActiveDocument.Utility.GetAngle method
getcfg	AutoCAD.Application.Preferences.property property
getcname	<i>Not provided</i>
getcorner	AutoCAD.Application.ActiveDocument.Utility.GetCorner method
getdist	AutoCAD.Application.ActiveDocument.Utility.GetDistance method
getenv	AutoCAD.Application.Preferences.property property
getfiled	Use Visual Basic file dialog
getint	AutoCAD.Application.ActiveDocument.Utility.GetInteger method
getkeyword	AutoCAD.Application.ActiveDocument.Utility.GetKeyword method
getorient	AutoCAD.Application.ActiveDocument.Utility.GetOrientation method

Paper space, model space, and TILEMODE settings (continued)

AutoLISP function	ActiveX or Visual Basic equivalent
getpoint	AutoCAD.Application.ActiveDocument.Utility.GetPoint method
getreal	AutoCAD.Application.ActiveDocument.Utility.GetReal method
getstring	AutoCAD.Application.ActiveDocument.Utility.GetString method
getvar	AutoCAD.Application.GetVariable method
graphscr	AppActivate AutoCAD.Application.Caption
grclear	<i>Obsolete function</i>
grdraw	<i>Not provided</i>
gread	<i>Not provided</i>
grtext	AutoCAD.Application.ActiveDocument.Utility.Prompt
grvecs	<i>Not provided</i>
handent	AutoCAD.Application.ActiveDocument.ModelSpace.object.Handle property
help	Search for HELP in the online Help index
if	If... Then... Else statement
initget	AutoCAD.Application.ActiveDocument.Utility.InitializeUserInput
inters	AutoCAD.Application.ActiveDocument.ModelSpace.object.IntersectWith
itoa	Str function
lambda	<i>Not provided</i>
last	arrayname(UBound(arrayname))
length	UBound function
list	ReDim statement

Paper space, model space, and TILEMODE settings (continued)

AutoLISP function	ActiveX or Visual Basic equivalent
listp	IsArray function
load_dialog	Use the Visual Basic Dialog Editor
load	AutoLISP is not supported through Automation
log	Log function
logand	And function
logior	Or function
lsh	Imp function
mapcar	<i>Not provided</i>
max	Max function
mem	<i>Not provided</i>
member	Use collection
menucmd	AutoCAD.Application.MenuBar object
menugroup	AutoCAD.Application.MenuGroup object
min	Min function
minusp	Use < 0 syntax
mode_tile	Use the Visual Basic Dialog Editor
namedobjdict	AutoCAD.Application.ActiveDocument.Dictionaries collection
nentsel	AutoCAD.Application.ActiveDocument.SelectionSets. SelectionSet.SelectAtPoint method
nentselp	AutoCAD.Application.ActiveDocument.SelectionSets. SelectionSet.SelectAtPoint method
new_dialog	Use the Visual Basic Dialog Editor
not	Use the logical operators
nth	Use object(n) syntax

Paper space, model space, and TILEMODE settings (continued)

AutoLISP function	ActiveX or Visual Basic equivalent
null	IsNull function
numberp	TypeName function
open	Open function
or	Use the logical operators
osnap	<i>Not provided</i> (You can use the SetVariable method to control the OSMODE system variable.)
polar	AutoCAD.Application.ActiveDocument.Utility.PolarPoint method
prin1	AutoCAD.Application.ActiveDocument.Utility.Prompt
princ	AutoCAD.Application.ActiveDocument.Utility.Prompt
print	AutoCAD.Application.ActiveDocument.Utility.Prompt
progn	<i>Not provided</i>
prompt	AutoCAD.Application.ActiveDocument.Utility.Prompt
quit	AutoCAD.Application.Quit method
quote	<i>Not provided</i>
read	<i>Not provided</i>
read-char	Input function
read-line	Line Input function
redraw	AutoCAD.Application.ActiveDocument.Modelspace.object.Update method
regapp	AutoCAD.Application.ActiveDocument.RegisteredApplications.Add method
rem	Mod function
repeat	For... Each, While,
reverse	<i>Not provided</i>

Paper space, model space, and TILEMODE settings (continued)

AutoLISP function	ActiveX or Visual Basic equivalent
rtos	AutoCAD.Application.ActiveDocument.Utility.RealToString method
set	Set function
set_tile	Use the Visual Basic Dialog Editor
setcfg	AutoCAD.Application.Preferences.property property
setfunhelp	<i>Not provided</i>
setq	Set function
setvar	AutoCAD.Application.SetVariable method
sin	Sin function
setview	AutoCAD.Application.ActiveDocument.Viewports.Viewport.SetView method
slide_image	Use the Visual Basic Dialog Editor
snvalid	<i>Not provided</i>
sqr	Sqr function
ssadd	AutoCAD.Application.ActiveDocument.SelectionSets.Add method
ssdel	AutoCAD.Application.ActiveDocument.SelectionSets.SelectionSet.Delete method
ssget	AutoCAD.Application.ActiveDocument.SelectionSets.SelectionSet.SelectOnScreen method
ssgetfirst	<i>Not provided</i>
sslength	AutoCAD.Application.ActiveDocument.SelectionSets.SelectionSet.Count method
ssmemb	Compare ID of object with the SelectionSet members
ssname	AutoCAD.Application.ActiveDocument.SelectionSets.SelectionSet.Name property
ssnamex	<i>Not provided</i>

Paper space, model space, and TILEMODE settings (continued)

AutoLISP function	ActiveX or Visual Basic equivalent
sssetfirst	AutoCAD.Application.ActiveDocument.PickfirstSelectionSet
startapp	Shell function
start_dialog	Use the Visual Basic Dialog Editor
start_image	Use the Visual Basic Dialog Editor
start_list	Use the Visual Basic Dialog Editor
strcase	StrConv function
strcat	& operator
strlen	Len function
subst	<i>Not provided</i>
substr	Mid function
tablet	<i>Not provided</i>
tblnext	AutoCAD.Application.ActiveDocument.collection_object. Item method
tblobjname	AutoCAD.Application.ActiveDocument.collection_object. Name method
tblsearch	AutoCAD.Application.ActiveDocument.collection_object. Name method
term_dialog	Use the Visual Basic Dialog Editor
terpri	<i>Not provided</i>
textbox	AutoCAD.Application.ActiveDocument.space.object. GetBoundingBox method
textpage	<i>Not provided</i>
textscr	<i>Not provided</i>
trace	<i>Not provided</i>
trans	AutoCAD.Application.ActiveDocument.Utility. TranslateCoordinates method

Paper space, model space, and TILEMODE settings (*continued*)

AutoLISP function	ActiveX or Visual Basic equivalent
type	TypeName function
unload_dialog	Use the Visual Basic Dialog Editor
untrace	<i>Not provided</i>
vector_image	Use the Visual Basic Dialog Editor
ver	AutoCAD.Application.Version property
vports	AutoCAD.Application.ActiveDocument.Viewports collection
wcmatch	Like operator
while	While... Wend
write-char	Print function
write-line	Print function
xdroom	<i>Not provided</i>
xdsize	<i>Not provided</i>
zerop	Use = 0 syntax

Migrating from AutoCAD Release 14.01

This appendix lists the new objects, methods, properties, and events for the ActiveX Automation interface since AutoCAD 14.01. It also lists the methods and properties that have changed and those that have been removed from the system.

B

In this appendix

- New Items
- Changed Items
- Removed Items

New Items

The following section lists all ActiveX objects along with their new methods, events, and properties.

This section lists all previously existing, new, and hidden objects in ActiveX. (Hidden objects are exposed as COM interfaces but not as VBA objects. The methods and properties on hidden objects are available to VBA users through other VBA objects.)

In the following table, new objects are identified with an asterisk (*) and hidden objects are identified with a plus sign (+).

New methods, properties, and events since AutoCAD Release 14.01

Object	Method/Property/Event name
3dFace	Coordinate, Coordinates, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, VisibilityEdge1, VisibilityEdge2, VisibilityEdge3, VisibilityEdge4
3dPolyline	Coordinate, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, Type
3dSolid	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Application	AppActivate, AppDeactivate, ARXLoaded, ARXUnloaded, BeginCommand, BeginFileDrop, BeginLisp, BeginModal, BeginOpen, BeginPlot, BeginQuit, BeginSave, Documents, EndCommand, EndLisp, EndModal, EndOpen, EndPlot, EndSave, Eval, GetAcadState, LispCancelled, LoadDVB, MenuBar, MenuGroups, NewDrawing, RunMacro, StatusId, SysVarChanged, UnloadDVB, VBE, WindowChanged, WindowLeft, WindowMovedOrResized, WindowState, WindowTop, ZoomAll, ZoomCenter, ZoomExtents, ZoomPickWindow, ZoomScaled, ZoomWindow, ZoomPrevious
Arc	ArcLength, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, TotalAngle
Attribute	Alignment, Backward, Constant, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Invisible, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, Preset, UpsideDown, Verify

New methods, properties, and events since AutoCAD Release 14.01 (continued)

Object	Method/Property/Event name
AttributeReference	Alignment, ArrayPolar, ArrayRectangular, Backward, Constant, Copy, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Invisible, Lineweight, Mirror, Mirror3D, Modified, ObjectName, OwnerID, PlotStyleName, UpsideDown
Block	AddDim3PointAngular, AddMInsertBlock, AddMLine, AddPolyfaceMesh, AttachExternalReference, Bind, Detach, Document, GetExtensionDictionary, GetXData, HasExtensionDictionary, IsLayout, IsXRef, Layout, Modified, ObjectID, ObjectName, OwnerID, Reload, SetXData, Unload, XRefDatabase
BlockReference	Delete, Document, GetConstantAttributes, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Blocks	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Circle	Circumference, Delete, Diameter, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Database*	Blocks, CopyFrom, CopyObjects, Dictionaries, DimStyles, ElevationModelspace, ElevationPaperspace, Groups, HandleToObject, Layers, Layouts, Limits, Linetypes, ModelSpace, ObjectIdToObject, PaperSpace, PlotConfigurations, Preferences, RegisteredApplications, TextStyles, UserCoordinateSystems, Viewports, Views
DatabasePreferences*	AllowLongSymbolNames, Application, ContourLinesPerSurface, DisplaySilhouette, Lineweight, LineWeightDisplay, MaxActiveViewports, ObjectSortByPlotting, ObjectSortByPSOutput, ObjectSortByRedraws, ObjectSortByRegens, ObjectSortBySelection, ObjectSortBySnap, OLELaunch, RenderSmoothness, SegmentPerPolyline, SolidFill, TextFrameDisplay, XRefEdit, XRefLayerVisibility
Dictionaries	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Dictionary	AddXRecord, Count, Document, GetExtensionDictionary, HasExtensionDictionary, Item, Modified, ObjectID, ObjectName, OwnerID

New methods, properties, and events since AutoCAD Release 14.01 (continued)

Object	Method/Property/Event name
Dim3PointAngular*	AngleFormat, AngleVertex, Application, ArrayPolar, ArrayRectangular, Arrowhead1Block, Arrowhead1Type, Arrowhead2Block, Arrowhead2Type, ArrowheadSize, Color, Copy, DecimalSeparator, Delete, DimensionLineColor, DimensionLineWeight, DimLine1Suppress, DimLine2Suppress, DimLineInside, Document, ExtensionLineColor, ExtensionLineExtend, ExtensionLineOffset, ExtensionLineWeight, ExtLine1EndPoint, ExtLine1Suppress, ExtLine2EndPoint, ExtLine2Suppress, Fit, ForceLineInside, GetBoundingBox, GetExtensionDictionary, GetXdata, Handle, HasExtensionDictionary, Highlight, HorizontalTextPosition, Hyperlinks, IntersectWith, Layer, Linetype, LinetypeScale, Lineweight, Measurement, Mirror, Mirror3D, Modified, Move, Normal, ObjectID, ObjectName, OwnerID, PlotStyleName, Rotate, Rotate3D, Rotation, ScaleEntity, ScaleFactor, SetXdata, StyleName, SuppressLeadingZeros, SuppressTrailingZeros, TextColor, TextGap, TextHeight, TextInside, TextInsideAlign, TextMovement, TextOutsideAlign, TextOverride, TextPosition, TextPrecision, TextPrefix, TextRotation, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceUpperLimit, TransformBy, Update, VerticalTextPosition, Visible
DimAligned	AltRoundDistance, AltSuppressLeadingZeros, AltSuppressTrailingZeros, AltSuppressZeroFeet, AltSuppressZeroInches, AltTextPrefix, AltTextSuffix, AltTolerancePrecision, AltToleranceSuppressLeadingZeros, AltToleranceSuppressTrailingZeros, AltToleranceSuppressZeroFeet, AltToleranceSuppressZeroInches, AltUnits, AltUnitsFormat, AltUnitsPrecision, AltUnitsScale, Arrowhead1Block, Arrowhead1Type, Arrowhead2Block, Arrowhead2Type, ArrowheadSize, DecimalSeparator, Delete, DimensionLineColor, DimensionLineExtend, DimensionLineWeight, DimLine1Suppress, DimLine2Suppress, DimLineInside, Document, ExtensionLineColor, ExtensionLineExtend, ExtensionLineOffset, ExtensionLineWeight, ExtLine1Suppress, ExtLine2Suppress, Fit, ForceLineInside, FractionFormat, GetExtensionDictionary, HasExtensionDictionary, HorizontalTextPosition, Hyperlinks, LinearScaleFactor, Lineweight, Measurement, Modified, ObjectName, OwnerID, PlotStyleName, PrimaryUnitsPrecision, RoundDistance, ScaleFactor, SuppressLeadingZeros, SuppressTrailingZeros, SuppressZeroFeet, SuppressZeroInches, TextColor, TextGap, TextHeight, TextInside, TextInsideAlign, TextMovement, TextOutsideAlign, TextOverride, TextPrefix, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceSuppressZeroFeet, ToleranceSuppressZeroInches, ToleranceUpperLimit, UnitsFormat, VerticalTextPosition

New methods, properties, and events since AutoCAD Release 14.01 (continued)

Object	Method/Property/Event name
DimAngular	AngleFormat, Arrowhead1Block, Arrowhead1Type, Arrowhead2Block, Arrowhead2Type, ArrowheadSize, DecimalSeparator, Delete, DimensionLineColor, DimensionLineWeight, DimLine1Suppress, DimLine2Suppress, DimLineInside, Document, ExtensionLineColor, ExtensionLineExtend, ExtensionLineOffset, ExtensionLineWeight, ExtLine1Suppress, ExtLine2Suppress, Fit, ForceLineInside, GetExtensionDictionary, HasExtensionDictionary, HorizontalTextPosition, Hyperlinks, Lineweight, Measurement, Modified, ObjectName, OwnerID, PlotStyleName, ScaleFactor, SuppressLeadingZeros, SuppressTrailingZeros, TextColor, TextGap, TextHeight, TextInside, TextInsideAlign, TextMovement, TextOutsideAlign, TextOverride, TextPrecision, TextPrefix, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceUpperLimit, VerticalTextPosition
DimDiametric	AltRoundDistance, AltSuppressLeadingZeros, AltSuppressTrailingZeros, AltSuppressZeroFeet, AltSuppressZeroInches, AltTextPrefix, AltTextSuffix, AltTolerancePrecision, AltToleranceSuppressLeadingZeros, AltToleranceSuppressTrailingZeros, AltToleranceSuppressZeroFeet, AltToleranceSuppressZeroInches, AltUnits, AltUnitsFormat, AltUnitsPrecision, AltUnitsScale, Arrowhead1Block, Arrowhead1Type, Arrowhead2Block, Arrowhead2Type, ArrowheadSize, CenterMarkSize, CenterType, DecimalSeparator, Delete, DimensionLineColor, DimensionLineWeight, DimLine1Suppress, DimLine2Suppress, Document, Fit, ForceLineInside, FractionFormat, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, LinearScaleFactor, Lineweight, Measurement, Modified, ObjectName, OwnerID, PlotStyleName, PrimaryUnitsPrecision, RoundDistance, ScaleFactor, SuppressLeadingZeros, SuppressTrailingZeros, SuppressZeroFeet, SuppressZeroInches, TextColor, TextGap, TextHeight, TextInside, TextInsideAlign, TextMovement, TextOutsideAlign, TextOverride, TextPrefix, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceSuppressZeroFeet, ToleranceSuppressZeroInches, ToleranceUpperLimit, UnitsFormat, VerticalTextPosition
Dimension+	Application, ArrayPolar, ArrayRectangular, Color, Copy, DecimalSeparator, Delete, Document, GetBoundingBox, GetExtensionDictionary, GetXdata, Handle, HasExtensionDictionary, Highlight, Hyperlinks, IntersectWith, Layer, Linetype, LinetypeScale, Lineweight, Mirror, Mirror3D, Modified, Move, Normal, ObjectID, ObjectName, OwnerID, PlotStyleName, Rotate, Rotate3D, Rotation, ScaleEntity, ScaleFactor, SetXdata, StyleName, SuppressLeadingZeros, SuppressTrailingZeros, TextColor, TextGap, TextHeight, TextMovement, TextOverride, TextPosition, TextPrefix, TextRotation, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceUpperLimit, TransformBy, Update, VerticalTextPosition, Visible

New methods, properties, and events since AutoCAD Release 14.01 (continued)

Object	Method/Property/Event name
DimOrdinate	AltRoundDistance, AltSuppressLeadingZeros, AltSuppressTrailingZeros, AltSuppressZeroFeet, AltSuppressZeroInches, AltTextPrefix, AltTextSuffix, AltTolerancePrecision, AltToleranceSuppressLeadingZeros, AltToleranceSuppressTrailingZeros, AltToleranceSuppressZeroFeet, AltToleranceSuppressZeroInches, AltUnits, AltUnitsFormat, AltUnitsPrecision, AltUnitsScale, ArrowheadSize, DecimalSeparator, Delete, Document, ExtensionLineColor, ExtensionLineOffset, ExtensionLineWeight, FractionFormat, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, LinearScaleFactor, Lineweight, Measurement, Modified, ObjectName, OwnerID, PlotStyleName, PrimaryUnitsPrecision, RoundDistance, ScaleFactor, SuppressLeadingZeros, SuppressTrailingZeros, SuppressZeroFeet, SuppressZeroInches, TextColor, TextGap, TextHeight, TextMovement, TextOverride, TextPrefix, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceSuppressZeroFeet, ToleranceSuppressZeroInches, ToleranceUpperLimit, UnitsFormat, VerticalTextPosition
DimRadial	AltRoundDistance, AltSuppressLeadingZeros, AltSuppressTrailingZeros, AltSuppressZeroFeet, AltSuppressZeroInches, AltTextPrefix, AltTextSuffix, AltTolerancePrecision, AltToleranceSuppressLeadingZeros, AltToleranceSuppressTrailingZeros, AltToleranceSuppressZeroFeet, AltToleranceSuppressZeroInches, AltUnits, AltUnitsFormat, AltUnitsPrecision, AltUnitsScale, ArrowheadBlock, ArrowheadSize, ArrowheadType, CenterMarkSize, CenterType, DecimalSeparator, Delete, DimensionLineColor, DimensionLineWeight, DimLineSuppress, Document, Fit, ForceLineInside, FractionFormat, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, LinearScaleFactor, Lineweight, Measurement, Modified, ObjectName, OwnerID, PlotStyleName, PrimaryUnitsPrecision, RoundDistance, ScaleFactor, SuppressLeadingZeros, SuppressTrailingZeros, SuppressZeroFeet, SuppressZeroInches, TextColor, TextGap, TextHeight, TextInside, TextInsideAlign, TextMovement, TextOutsideAlign, TextOverride, TextPrefix, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceSuppressZeroFeet, ToleranceSuppressZeroInches, ToleranceUpperLimit, UnitsFormat, VerticalTextPosition

New methods, properties, and events since AutoCAD Release 14.01 (continued)

Object	Method/Property/Event name
DimRotated	AltRoundDistance, AltSuppressLeadingZeros, AltSuppressTrailingZeros, AltSuppressZeroFeet, AltSuppressZeroInches, AltTextPrefix, AltTextSuffix, AltTolerancePrecision, AltToleranceSuppressLeadingZeros, AltToleranceSuppressTrailingZeros, AltToleranceSuppressZeroFeet, AltToleranceSuppressZeroInches, AltUnits, AltUnitsFormat, AltUnitsPrecision, AltUnitsScale, Arrowhead1Block, Arrowhead1Type, Arrowhead2Block, Arrowhead2Type, ArrowheadSize, DecimalSeparator, Delete, DimensionLineColor, DimensionLineExtend, DimensionLineWeight, DimLine1Suppress, DimLine2Suppress, DimLineInside, Document, ExtensionLineColor, ExtensionLineExtend, ExtensionLineOffset, ExtensionLineWeight, ExtLine1Suppress, ExtLine2Suppress, Fit, ForceLineInside, FractionFormat, GetExtensionDictionary, HasExtensionDictionary, HorizontalTextPosition, Hyperlinks, LinearScaleFactor, Lineweight, Measurement, Modified, ObjectName, OwnerID, PlotStyleName, PrimaryUnitsPrecision, RoundDistance, ScaleFactor, SuppressLeadingZeros, SuppressTrailingZeros, SuppressZeroFeet, SuppressZeroInches, TextColor, TextGap, TextHeight, TextInside, TextInsideAlign, TextMovement, TextOutsideAlign, TextOverride, TextPrefix, TextStyle, TextSuffix, ToleranceDisplay, ToleranceHeightScale, ToleranceJustification, ToleranceLowerLimit, TolerancePrecision, ToleranceSuppressLeadingZeros, ToleranceSuppressTrailingZeros, ToleranceSuppressZeroFeet, ToleranceSuppressZeroInches, ToleranceUpperLimit, UnitsFormat, VerticalTextPosition
DimStyle	CopyFrom, Document, GetExtensionDictionary, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID
DimStyles	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Document	Activate, Active, ActiveLayout, BeginClose, BeginDoubleClick, BeginLisp, BeginPlot, BeginRightClick, BeginShortcutMenuCommand, BeginShortcutMenuDefault, BeginShortcutMenuEdit, BeginShortcutMenuGrip, BeginShortcutMenuOsnap, Close, CopyObjects, Database, Deactivate, EndLisp, EndPlot, EndShortcutMenu, EndUndoMark, Height, HWND, Layouts, LayoutSwitched, LispCancelled, ObjectAdded, ObjectErased, ObjectModified, PickfirstSelectionSet, PlotConfigurations, Preferences, SelectionChanged, SendCommand, StartUndoMark, Width, WindowChanged, WindowMovedOrResized, WindowState, WindowTitle
Documents*	Add, Application, Close, Count, Item, Open
Ellipse	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, MajorRadius, MinorRadius, Modified, ObjectName, OwnerID, PlotStyleName
Entity+	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName

New methods, properties, and events since AutoCAD Release 14.01 (continued)

Object	Method/Property/Event name
ExternalReference*	Application, ArrayPolar, ArrayRectangular, Color, Copy, Delete, Document, Explode, GetAttributes, GetBoundingBox, GetConstantAttributes, GetExtensionDictionary, GetXdata, Handle, HasAttributes, HasExtensionDictionary, Highlight, Hyperlinks, InsertionPoint, IntersectWith, Layer, Linetype, LinetypeScale, Lineweight, Mirror, Mirror3D, Modified, Move, Name, Normal, ObjectID, ObjectName, OwnerID, Path, PlotStyleName, Rotate, Rotate3D, Rotation, ScaleEntity, SetXdata, TransformBy, Update, Visible, XScaleFactor, YScaleFactor, ZScaleFactor
Group	Document, GetExtensionDictionary, HasExtensionDictionary, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Groups	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Hatch	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, ISOPenWidth, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Hyperlink*	Application, Delete, URL, URLDescription, URLNamedLocation
Hyperlinks*	Add, Application, Count, Item
IDPair*	Application, IsCloned, IsOwnerXlated, IsPrimary, Key, Value
Layer	Document, GetExtensionDictionary, HasExtensionDictionary, Lineweight, Modified, ObjectID, ObjectName, OwnerID, PlotStyleName, Plottable, ViewportDefault
Layers	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
LayerStateManager	Delete, Export, Import, Mask, Rename, Restore, Save, SetDatabase
Layout*	Application, Block, CanonicalMediaName, CenterPlot, ConfigName, CopyFrom, Delete, Document, GetCanonicalMediaNames, GetCustomScale, GetExtensionDictionary, GetLocaleMediaName, GetPaperMargins, GetPaperSize, GetPlotDeviceNames, GetPlotStyleTableNames, GetWindowToPlot, GetXdata, Handle, HasExtensionDictionary, ModelType, Modified, Name, ObjectID, ObjectName, OwnerID, PaperUnits, PlotHidden, PlotOrigin, PlotRotation, PlotType, PlotViewportBorders, PlotViewportsFirst, PlotWithLineweights, PlotWithPlotStyles, RefreshPlotDeviceInfo, ScaleLineweights, SetCustomScale, SetWindowToPlot, SetXdata, ShowPlotStyles, StandardScale, StyleSheet, TabOrder, UseStandardScale, ViewToPlot

New methods, properties, and events since AutoCAD Release 14.01 (continued)

Object	Method/Property/Event name
Layouts*	Add, Application, Count, Delete, Document, GetExtensionDictionary, GetXdata, Handle, HasExtensionDictionary, Item, Modified, ObjectID, ObjectName, OwnerID, SetXdata
Leader	Annotation, ArrowheadBlock, ArrowheadSize, ArrowheadType, Coordinate, Delete, DimensionLineColor, DimensionLineWeight, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, ScaleFactor, TextGap, VerticalTextPosition
Line	Angle, Delete, Delta, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Length, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Linetype	Document, GetExtensionDictionary, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID
Linetypes	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
LightweightPolyline	ConstantWidth, Coordinate, Delete, Document, Elevation, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, LinetypeGeneration, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
MenuBar*	Application, Count, Item, Parent
MenuGroup*	Application, MenuFileName, Menus, Name, Parent, Save, SaveAs, Toolbars, Type, Unload
MenuGroups*	Application, Count, Item, Load, Parent
MInsertBlock*	Application, ArrayPolar, ArrayRectangular, Color, Columns, ColumnSpacing, Copy, Delete, Document, Explode, GetAttributes, GetBoundingBox, GetConstantAttributes, GetExtensionDictionary, GetXdata, Handle, HasAttributes, HasExtensionDictionary, Highlight, Hyperlinks, InsertionPoint, IntersectWith, Layer, Linetype, LinetypeScale, Lineweight, Mirror, Mirror3D, Modified, Move, Name, Normal, ObjectID, ObjectName, OwnerID, PlotStyleName, Rotate, Rotate3D, Rotation, Rows, RowSpacing, ScaleEntity, SetXdata, TransformBy, Update, Visible, XScaleFactor, YScaleFactor, ZScaleFactor
MLine*	Application, ArrayPolar, ArrayRectangular, Color, Coordinates, Copy, Delete, Document, GetBoundingBox, GetExtensionDictionary, GetXdata, Handle, HasExtensionDictionary, Highlight, Hyperlinks, IntersectWith, Layer, Linetype, LinetypeScale, Lineweight, Mirror, Mirror3D, Modified, Move, ObjectID, ObjectName, OwnerID, PlotStyleName, Rotate, Rotate3D, ScaleEntity, SetXdata, StyleName, TransformBy, Update, Visible

New methods, properties, and events since AutoCAD Release 14.01 (continued)

Object	Method/Property/Event name
ModelSpace	AddDim3PointAngular, AddMInsertBlock, AddMLine, AddPolyfaceMesh, AttachExternalReference, Bind, Delete, Detach, Document, GetExtensionDictionary, GetXData, HasExtensionDictionary, IsLayout, IsXRef, Layout, Modified, ObjectID, ObjectName, Origin, OwnerID, Reload, SetXData, Unload, XRefDatabase
MText	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, LineSpacingFactor, LineSpacingStyle, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Object+	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Modified, ObjectName, OwnerID
PaperSpace	AddDim3PointAngular, AddMInsertBlock, AddMLine, AddPolyfaceMesh, AttachExternalReference, Bind, Delete, Detach, Document, GetExtensionDictionary, GetXData, HasExtensionDictionary, IsLayout, IsXRef, Layout, Modified, ObjectID, ObjectName, Origin, OwnerID, Reload, SetXData, Unload, XRefDatabase
Plot	BatchPlotProgress, DisplayPlotPreview, NumberOfCopies, QuietErrorMode, SetLayoutsToPlot, StartBatchMode
PlotConfiguration*	Application, CanonicalMediaName, CenterPlot, ConfigName, CopyFrom, Delete, Document, GetCanonicalMediaNames, GetCustomScale, GetExtensionDictionary, GetLocaleMediaName, GetPaperMargins, GetPaperSize, GetPlotDeviceNames, GetPlotStyleTableNames, GetWindowToPlot, GetXdata, Handle, HasExtensionDictionary, ModelType, Modified, Name, ObjectID, ObjectName, OwnerID, PaperUnits, PlotHidden, PlotOrigin, PlotRotation, PlotType, PlotViewportBorders, PlotViewportsFirst, PlotWithLineweights, PlotWithPlotStyles, RefreshPlotDeviceInfo, ScaleLineweights, SetCustomScale, SetWindowToPlot, SetXdata, ShowPlotStyles, StandardScale, StyleSheet, UseStandardScale, ViewToPlot
PlotConfigurations*	Add, Application, Count, Delete, Document, GetExtensionDictionary, GetXdata, Handle, HasExtensionDictionary, Item, Modified, ObjectID, ObjectName, OwnerID, SetXdata
Point	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
PolyfaceMesh*	Application, ArrayPolar, ArrayRectangular, Color, Coordinate, Coordinates, Copy, Delete, Document, GetBoundingBox, GetExtensionDictionary, GetXdata, Handle, HasExtensionDictionary, Highlight, Hyperlinks, IntersectWith, Layer, Linetype, LinetypeScale, Lineweight, Mirror, Mirror3D, Modified, Move, NumberOfFaces, NumberOfVertices, ObjectID, ObjectName, OwnerID, PlotStyleName, Rotate, Rotate3D, ScaleEntity, SetXdata, TransformBy, Update, Visible

New methods, properties, and events since AutoCAD Release 14.01 (continued)

Object	Method/Property/Event name
PolygonMesh	Coordinate, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Polyline	ConstantWidth, Coordinate, Delete, Document, Elevation, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, LinetypeGeneration, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
PopupMenu*	AddMenuItem, AddSeparator, AddSubMenu, Application, Count, InsertInMenuBar, Item, Name, NameNoMnemonic, OnMenuBar, Parent, RemoveFromMenuBar, ShortcutMenu, TagString
PopupMenuItem*	Application, Caption, Check, Delete, Enable, HelpString, Index, Label, Macro, Parent, SubMenu, TagString, Type
PopupMenu*	Add, Application, Count, InsertMenuInMenuBar, Item, Parent, RemoveMenuFromMenuBar
Preferences	Display, Drafting, Files, OpenSave, Output, Profiles, Selection, System, User
PreferencesDisplay*	Application, AutoTrackingVecColor, CursorSize, DisplayLayoutTabs, DisplayScreenMenu, DisplayScrollBars, DockedVisibleLines, GraphicsWinLayoutBackgrndColor, GraphicsWinModelBackgrndColor, HistoryLines, ImageFrameHighlight, LayoutCreateViewport, LayoutCrosshairColor, LayoutDisplayMargins, LayoutDisplayPaper, LayoutDisplayPaperShadow, LayoutShowPlotSetup, MaxAutoCADWindow, ModelCrosshairColor, ShowRasterImage, TextFont, TextFontSize, TextFontStyle, TextWinBackgrndColor, TextWinTextColor, TrueColorImages, XRefFadeIntensity
PreferencesDrafting*	AlignmentPointAcquisition, Application, AutoSnapAperture, AutoSnapApertureSize, AutoSnapMagnet, AutoSnapMarker, AutoSnapMarkerColor, AutoSnapMarkerSize, AutoSnapTooltip, AutoTrackTooltip, FullScreenTrackingVector, PolarTrackingVector
PreferencesFiles*	AltFontFile, AltTabletMenuFile, Application, AutoSavePath, ConfigFile, CustomDictionary, DefaultInternetURL, DriversPath, FontFileMap, GetProjectFilePath, HelpFilePath, LicenseServer, LogFilePath, MainDictionary, MenuFile, ObjectARXPath, PostScriptPrologFile, PrinterConfigPath, PrinterDescPath, PrinterStyleSheetPath, PrintFile, PrintSpoolerPath, PrintSpoolExecutable, SetProjectFilePath, SupportPath, TempFilePath, TemplateDwgPath, TempXrefPath, TextEditor, TextureMapPath, WorkspacePath
PreferencesOpenSave*	Application, AutoAudit, AutoSaveInterval, CreateBackup, DemandLoadArxApp, FullCrcValidation, IncrementalSavePercent, LogFileOn, MRUNumber, ProxyImage, SaveAsType, SavePreviewThumbnail, ShowProxyDialogBox, TempFileExtension, XrefDemandLoad

New methods, properties, and events since AutoCAD Release 14.01 (continued)

Object	Method/Property/Event name
PreferencesOutput*	Application, DefaultOutputDevice, DefaultPlotStyleForLayer, DefaultPlotStyleForObjects, DefaultPlotStyleTable, OLEQuality, PlotLegacy, PlotPolicy, PrinterPaperSizeAlert, PrinterSpoolAlert, UseLastPlotSettings
PreferencesProfiles*	ActiveProfile, Application, CopyProfile, DeleteProfile, ExportProfile, GetAllProfileNames, ImportProfile, RenameProfile, ResetProfile
PreferencesSelection*	Application, DisplayGrips, DisplayGripsWithinBlocks, GripColorSelected, GripColorUnselected, GripSize, PickAdd, PickAuto, PickBoxSize, PickDrag, PickFirst, PickGroup
PreferencesSystem*	Application, BeepOnError, DisplayOLEScale, EnableStartupDialog, LoadAcadLspInAllDocuments, ShowWarningMessages, SingleDocumentMode, StoreSQLIndex, TablesReadOnly
PreferencesUser*	ADCInsertUnitsDefaultSource, ADCInsertUnitsDefaultTarget, Application, HyperlinkDisplayCursor, HyperlinkDisplayTooltip, KeyboardAccelerator, KeyboardPriority, SCMCommandMode, SCMDefaultMode, SCMEditMode, ShortCutMenuDisplay
PViewport	ArcSmoothness, Clipped, CustomScale, Delete, DisplayLocked, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, StandardScale, StyleSheet, UCSPerViewport, ViewportOn
Raster	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, ImageHeight, ImageWidth, Lineweight, Modified, Name, ObjectName, OwnerID, PlotStyleName, Rotation, ScaleFactor, ShowRotation
Ray	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, SecondPoint
Region	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
RegisteredApplication	Document, GetExtensionDictionary, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID
RegisteredApplications	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Shape	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
Solid	Coordinate, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName

New methods, properties, and events since AutoCAD Release 14.01 (continued)

Object	Method/Property/Event name
Spline	ControlPoints, Delete, Document, FitPoints, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, IsPeriodic, IsPlanar, Knots, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, Weights
State*	Application, IsQuiescent
Text	Alignment, Backward, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, UpsideDown
TextStyle	Document, GetExtensionDictionary, GetFont, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetFont
TextStyles	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Tolerance	Delete, DimensionLineColor, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, ScaleFactor, TextColor, TextHeight, TextStyle
Toolbar*	AddSeparator, AddToolbarButton, Application, Count, Delete, Dock, DockStatus, Float, FloatingRows, Height, HelpString, Item, LargeButtons, Left, Name, Parent, TagString, Top, Visible, Width
ToolbarItem*	Application, AttachToolbarToFlyout, Delete, Flyout, GetBitmaps, HelpString, Index, Macro, Name, Parent, SetBitmaps, TagString, Type
Toolbars*	Add, Application, Count, Item, LargeButtons, Parent
Trace	Coordinate, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName
UCS	Document, GetExtensionDictionary, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID
UCSs	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Utility	GetRemoteFile, GetSubEntity, IsRemoteFile, IsURL, LaunchBrowserDialog, Prompt, PutRemoteFile
View	Document, GetExtensionDictionary, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID
Viewport	ArcSmoothness, Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID

New methods, properties, and events since AutoCAD Release 14.01 (continued)

Object	Method/Property/Event name
Viewports	DeleteConfiguration, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
Views	Delete, Document, GetExtensionDictionary, GetXData, Handle, HasExtensionDictionary, Modified, ObjectID, ObjectName, OwnerID, SetXData
XLine	Delete, Document, GetExtensionDictionary, HasExtensionDictionary, Hyperlinks, Lineweight, Modified, ObjectName, OwnerID, PlotStyleName, SecondPoint
XRecord*	Application, Delete, Document, GetExtensionDictionary, GetXdata, GetXRecordData, Handle, HasExtensionDictionary, Modified, Name, ObjectID, ObjectName, OwnerID, SetXdata, SetXRecordData, TranslateIDs

Changed Items

The following section briefly describes the existing methods and properties that have been changed since Release 14.01; see also “Changes to the Preferences Object” on page 362.

Changed methods and properties

Release 14 method/ property name	New AutoCAD signature	Description of changes
ArcSmoothness	object.ArcSmoothness	Moved from the Preferences object to the PViewport and Viewport object.
AuditInfo	object.AuditInfo FixError	Removed the Filename parameter.
AutoSaveFile	object.AutoSavePath	The name of this property has been changed to AutoSavePath. This property has also moved from the Preferences object to the PreferencesFiles object.
EndUndoMark	object.EndUndoMark	Moved from the Application object to the Document object.
GetUCSMatrix	RetVal = object.GetUCSMatrix()	Incorrectly performed a transpose operation when constructing the matrix. This method is now correctly row based.

Changed methods and properties (continued)

Release 14 method/ property name	New AutoCAD signature	Description of changes
InsertBlock	RetVal = object.InsertBlock(InsertionPoint, Name, Xscale, Yscale, ZScale, Rotation)	The ZScale parameter has been added.
LogFileName	object.LogFilePath	The name of this property has been changed to LogFilePath. This property has also moved from the Preferences object to the PreferencesFiles object.
StartUndoMark	object.StartUndoMark	Moved from the Application object to the Document object.
TextString (on Dimension objects)	object.TextOverride	The name of this property has been changed to TextOverride.
TransformBy	object.TransformBy TransformationMatrix	Incorrectly performed a transpose operation when constructing the matrix. This method is now correctly row based.
TranslateCoordinates	RetVal = TranslateCoordinates (OriginalPoint, From, To, Disp[,OCSNormal])	Added the OCSNormal parameter as well as the acOCS option for the From and To parameter to allow coordinate translations to and from an OCS.
ZoomAll	object.ZoomAll ()	This method has moved off the PViewport and Viewport objects to the Application object.
ZoomCenter	object.ZoomCenter Center, Magnify	This method has moved off the PViewport and Viewport objects to the Application object.
ZoomExtents	object.ZoomExtents	This method has moved off the PViewport and Viewport objects to the Application object.
ZoomPickWindow	object.ZoomPickWindow	This method has moved off the PViewport and Viewport objects to the Application object.
ZoomScaled	object.ZoomScaled Scale, ScaleType	This method has moved off the PViewport and Viewport objects to the Application object.

Changed methods and properties (continued)

Release 14 method/ property name	New AutoCAD signature	Description of changes
ZoomWindow	object.ZoomWindow LowerLeft, UpperRight	This method has moved off the PViewport and Viewport objects to the Application object.

Changes to the Preferences Object

The Preferences object has several major changes:

- The Preferences object has been subdivided extensively. The new objects that contain preferences are:
 - PreferencesDisplay
 - PreferencesDrafting
 - PreferencesFiles
 - PreferencesOpenSave
 - PreferencesOutput
 - PreferencesProfiles
 - PreferencesSelection
 - PreferencesSystem
 - PreferencesUser
 - DatabasePreferences

All properties and methods previously found on the Preferences object have been moved to one of the objects listed above. To find the new object for any given Preferences methods and properties, look up the method or property in the *AutoCAD ActiveX and VBA Reference*.

All properties specifying color have been changed to use the VB constant OLE_COLOR instead of the AutoCAD color constants. (Note that this applies only to properties found on the Preferences objects listed above.)

Removed Items

The following section briefly describes the existing methods and properties that have been removed since AutoCAD Release 14.01.

Removed methods and properties

Release 14 method/ property name	Migration options
AdjustAreaFill	Set in a PC3 configuration file (via the Plotter Configuration Editor).
CrosshairColor	Use PreferencesDisplay.ModelSpaceCrosshairColor or PreferencesDisplay.LayoutCrosshairColor.
EntityName	Use the AutoCAD ActiveX ObjectName property, or in VB and VBA use the TypeOf keyword or TypeName function.
EntityType	See EntityName.
Erase	Removed from all objects except Selection Set. Use the Delete method instead of Erase.
GraphicsTextBackgrndColor	Graphics text uses the same background color as the graphics. Use PreferencesDisplay.GraphicsWinLayoutBackgrndColor and PreferencesDisplay.GraphicsWinModelBackgrndColor.
GraphicsTextColor	Graphics text uses the same color as the pointer (crosshair). Use PreferencesDisplay.LayoutCrosshairColor and PreferencesDisplay.ModelCrosshairColor.
HideLines	Use AcadLayout.Hidden.
ListADS	EXE-based ADSRX apps are no longer supported. The method for querying ARX and DBX apps is ListARX.
LoadADS	EXE-based ADSRX apps are no longer supported. The method for loading ARX and DBX apps is LoadARX.
LoadPC2	PC2 files are no longer supported. You will need to convert your PC2 files to PC3 files via the Add-a-Plotter Wizard.
MonochromeVectors	Obsolete functionality.
Origin	Use AcadLayout.PlotOffset.
PaperSize	Use AcadLayout.CanonicalMediaName.

Removed methods and properties *(continued)*

Release 14 method/ property name	Migration options
PlotExtents	Use AcadLayout.AreaToPlot.
PlotLimits	Use AcadLayout.AreaToPlot.
PlotOrientation	Use AcadLayout.Rotation.
PlotScale	Use AcadLayout.Scale, StandardScale, CustomScale.
PlotUnits	Use AcadLayout.PaperUnits.
PlotView	Use AcadLayout.AreaToPlot and ViewToPlot.
PlotWindow	Use AcadLayout.AreaToPlot.
PlotWithConfigFile	Use PlotToDevice instead, and specify a PC3 file as an argument.
Rotation	Use AcadLayout.Rotation.
SavePC2	PC2 files are no longer supported. You will need to convert your PC2 files to PC3 files via the Add-a-Plotter Wizard.
UnloadADS	EXE-based ADSRX apps are no longer supported. The method for unloading ARX and DBX apps is UnloadARX.

Index

- \ (backslash)
 - format code, 165
 - ^ (caret)
 - stacked text indicator, 165
 - # (pound sign)
 - stacked text indicator, 165
 - / (slash, forward)
 - stacked text indicator, 165
 - 2D objects
 - editing, 107
 - positioning, 251
 - 3D modeling
 - 3DSolid object, 255
 - Add 3DPoly method, 252
 - mirroring (illustration), 258
 - rectangular arrays (illustration), 258
 - rotating (illustration), 256
 - solids
 - analyzing properties (list), 255
 - combining, 260
 - methods for creating (list), 255
 - wireframes, creating, 251
 - 3D polylines
 - creating, 251
 - 3DMesh object
 - example code, 252
 - 3DSolid object
 - defined, 255
 - example code, 255, 257, 259
- A**
- acad.dvb* project file, 13
 - ACAD_LAYERSTATE dictionary, 147
 - ACADProject* file, 15
 - acAttributeModeConstant constant, 294
 - acAttributeModeInvisible constant, 294
 - acAttributeModeNormal constant, 294
 - acAttributeModePreset constant, 295
 - acAttributeModeVerify constant, 295
 - acByBlock constant, 139
 - acByLayer constant, 139
 - accelerator keys, specifying for menus, 206
 - acDisplay, 266
 - acExtents, 266
 - ACI number, layers, 138
 - acIntersection constant, 92
 - acLayout, 266
 - acLimits, 266
 - acSubstraction constant, 92
 - Activate event, 237
 - Active UCS property, 67
 - ActiveDimStyle property, 182
 - ActiveLayer property, example code, 136
 - ActiveLinetype property, 141
 - ActivePViewport property, example code, 271
 - ActiveSpace property
 - example code, 270
 - view changing, 268, 269
 - ActiveTextStyle property, 67
 - ActiveUCS property, User Coordinate System, 248
 - ActiveViewport property
 - current viewport, 64
 - example code, 67, 252
 - resetting objects, 67
 - ActiveX interface
 - advantages with AutoCAD, 2
 - and AutoCAD objects, 4
 - and VBA applications, 4
 - types of objects, 2

- acUnion constant, 92
- acView, 266
- acWindow, 266
- Add method
 - blocks, example code, 291
 - collections, 38, 43
 - dimension styles, 183
 - Documents collection, 54
 - example code, 54
 - Layers collection, 134
 - layers, example code, 108, 134
 - leader lines, 191
 - menu groups, example code, 200
 - menus, example code, 203
 - selection sets, example code, 96
 - text styles, 153
 - toolbars, example code, 213
 - UCS, example code, 248
 - User Coordinate System, 247
- Add3DMesh method
 - example code, 252
 - MClose property, 252
 - NClose property, 252
- Add3DPoly method, 3D modeling, 252
- AddArc method, 86
- AddAttribute method
 - defining blocks, 294
- AddBox method
 - creating solids, 255
 - example code, 257, 259, 260
- AddCircle method
 - creating circles, 86
 - example code, 90
- AddCone method, 255
- AddCylinder method
 - creating solids, 255
 - example code, 260
- AddDimAligned method
 - example code, 182, 188
 - linear dimensions, 176
- AddDimAngular method
 - angular dimensions, 178
 - example code, 179
- AddDimDiametric method, 177
- AddDimOrdinate method
 - example code, 180
 - ordinate dimensions, 180
- AddDimRadial method, 177
 - example code, 178
 - radial dimensions, 177
- AddDimRotated method, 176
- AddEllipse method, 86
- AddEllipticalCone method, 255
- AddEllipticalCylinder method, 255
- AddExtrudedSolid method, 255
- AddExtrudedSolidAlongPath method, 255
- AddFitPoint method, splines, 128
- AddHatch method
 - creating objects, 92
 - example code, 94, 130, 132
- AddItem method, 96
- AddLeader method
 - creating leader lines, 190
 - example code, 190, 191
- AddLightweightPolyline method
 - example code, 85, 111
- AddLine method
 - example code, 122, 124
 - object hierarchy, 40
- AddMenuItem method, 204, 205
 - Count property, 204
 - example code, 200
- AddMInsertBlock method, 290
- AddMText method
 - example code, 164, 167, 191
 - multiline text, 163
- AddPoint method, example code, 88
- AddPolyfaceMesh method
 - creating meshes, 253
 - example code, 254
- AddPViewport method
 - example code, 271
 - paper space, 271
- AddRaster method
 - attaching images, 282
 - example code, 283, 288
- AddRegion method, 89
 - calculating total number of regions, 89
 - example code, 90
- AddRevolvedSolid method, 255
- AddSeparator method, using Type property, 206
- AddSolid method
 - creating solids, 255
 - example code, 88
 - solid-filled area, 88
- AddSphere method, 255
- AddSpline method
 - example code, 86, 129
 - NURBS, 86
- AddSubMenu method, example code, 208
- AddText method
 - example code, 157
- AddTolerance method
 - example code, 193
 - geometric tolerance, 193
- AddToolBarButton method, 213
 - creating flyout toolbars, 217
 - example code, 214, 217
- AddTorus method, 255
- AddVertex property, polylines, 126
- AddWedge method
 - creating solids, 255
 - example code, 255
- AddXLine method, in 3D, 70

- AddXLine object, example code, 70
- Alignment property, 160, 296
 - example code, 161
 - in text, 161
- AltFontFile property, 170
- AngleFromXAxis method, 72
- AngleToReal method, 72
- AngleToString method, 72
- angular dimensions
 - creating, 178
 - illustration, 172
- annotations, and leader lines, 189
- AppActivate event, 234
- AppDeactivate event, 234
- AppendInnerLoop method
 - boundaries, 93
 - example code, 130
 - hatches, 130
- AppendOuterLoop method
 - boundaries, 93
 - example code, 94, 130, 132
 - hatches, 130
- application level events
 - declaring objects with events, 236
 - enabling, 234
 - events (list), 234
- Application object
 - collections, accessing, 43
 - declaring objects with events, 236
 - events (list), 234
 - example code, 236, 330
 - root object, accessing, 36
- Application window
 - changing visibility, 36
 - finding state, 57
 - finding status, example code, 57
 - modifying, 36
 - sizing, 57
 - Visible property, 58
- application-level events
 - and VBA keywords, 235
 - enabling, application-level events
 - class modules, 235
- applications
 - declaring Windows APIs, 332
 - distribution options, 324
 - exchanging data with AutoCAD, 326
 - instances, creating, 328
 - object model referencing, 326
 - quit methods, 329
 - referencing application objects, 329
 - variables, declaring, 328
- Arc object
 - creating, 86
- Area property, 91
 - calculating for closed objects, 73
 - example code, 74, 90
 - splines, 128
- array data, converting to variants, 47
- arraying, patterns, 108
- ArrayPolar method
 - attributes, 297
 - creating arrays, 113
 - dimension editing, 181
 - Text object, 163
- ArrayRectangular method, 115, 258
 - attributes, 297
 - dimension editing, 181
 - Text object, 163
- arrays
 - polar, 113
 - rectangular, 113, 115
- ARXLoaded event, 234
- ARXUnloaded event, 234
- associative dimensions, defined, 175
- AssociativeHatch property, 129
- AttachExternalReference method
 - example code, 302, 304, 305
- AttachmentPoint property, 164
- AttachToolbarToFlyout method, example code, 217
- Attribute object
 - editing, 296
 - example code, 299
- attributes
 - associating, 294
 - in blocks (illustration), 294
 - constants (list), 294
 - creating references, 294
 - and definitions, 299
 - editing methods (list), 297
 - editing properties (list), 296
 - extracting, 294
 - and references, 299
 - for text, 294
 - visibility, 294
- Auto Data Tips, Code window, 28
- auto embedding, setting for VBA projects, 19
- Auto Indent, Code window, 28
- Auto List Member, Code window, 28
- Auto Quick Info, Code window, 28
- Auto Syntax Check, Code window, 28
- AutoCAD
 - exchanging data, 326
 - reinstalling, 5
- AutoCAD VBA, system requirements, 5

B

- backslash (\)
 - format code, 165
 - in macros, 225
- Backward property
 - attributes, 296
- base point, rotating objects, 117
- BasePoint property, 70
- BatchPlotProgress property, 277
- BeginClose event, 237
- BeginCommand event, 232, 234, 237
- BeginDoubleClick event, 238
- BeginFileDrop event
 - defined, 234
 - example code, 236
- BeginLISP event, 234, 238
- BeginModal event, 234
- BeginOpen event, 232, 234
- BeginPlot event, 234, 238
- BeginQuit event, 234
- BeginRightClick event, 238
- BeginSave event, 234, 238
- BeginShortcutMenuCommand event, 238
- BeginShortcutMenuDefault event
 - defined, 238
 - example code, 241
- BeginShortcutMenuEdit event, 238
- BeginShortcutMenuGrip event, 238
- BeginShortcutMenuOsnap event, 238
- Big Font files, 156
- BigFontFile property
 - example code, 156
 - TextStyle object, 154
- Bind method
 - example code, 307
 - external references, 306
- bitonal images. *See* raster images
- Block object
 - example code, 291, 292
 - in layouts, 265
- Block property, in layouts, 265
- block references
 - and arrays, 290
 - base points, 290
 - exploding, 289, 291
 - fastener (illustration), 290
 - iteration, 290
 - listing attributes, 330
 - naming, 290
 - redefining, 290
- BlockReference object
 - example code, 292, 299, 330
 - leader lines, 191

blocks

- attributes, 294
- binding, 306
- creating new blocks, 289
- creating objects, 84
- defined, 289
- exploding, 124
- inserting, 290
- and layers, 139, 144
- nesting (illustration), 289
- organizing objects, 289
- redefining, 293
- referencing in ActiveX code, 84

BMP files

- exporting to, 80
- importing, 80

Boolean intersection (illustration), 260

Boolean method, 260

- calculating area, 73
- composite regions, 90
- creating regions, 90
- example code, 90

boundaries, Hatch object, 92

Boundary Hatch dialog box

- pattern option, 131

boxes

- adding, example code, 257, 260
- mirroring, example code, 259
- slicing, example code, 262

Break mode, defined in VBA, 19

break on errors, for VBA projects, 19

Brightness property, 287

C

calculations

- AddRegion method, 89
- performing in drawings, 67

CanonicalMediaName property, 265

caret (^)

- stacked text indicator, 165

cascading menus. *See* submenus

CELTSCALE system variable, 142

Center property

- example code, 272
- WCS coordinate, 45

character formatting, example code, 167

characters, adjusting spaces between, 166

CheckInterference method, 260

- example code, 260

child object

- relation to root object, 45

Circle object, 45

- creating, 86
- example code, 90, 109

- circular references, 27
- Clear method
 - in selection sets, 106
- ClipBoundary method, 287
 - example code, 288
- ClippingEnabled property, 285, 287
 - example code, 288
- Close method
 - Documents collection, 54
- Closed property
 - polylines, 126
 - splines, 127
- Code window
 - margin indicator bar, 24
 - resizing, 24
 - settings, 28
 - split bar, 24
- collections
 - adding members, 38
 - Application object, accessing, 43
 - available in AutoCAD (list), 42
 - Count Property, 38
 - defined, 42
 - Document object, accessing, 43
 - Documents, 36
 - Item method, 38
 - MenuBar, 196
 - MenuGroup, 196
 - as object groups, 38
- Color property
 - changing linetypes, 144
 - example code, 97
 - layers, 138, 140
 - raster images, 285
- colors
 - assigning to objects, 143
- command line
 - controlling user input, 77
 - in zero document state, 323
 - prompting user, 75
- compile time
 - errors in VBA, 318
 - object referencing, 46
- ConfigName property, 267, 278
- configuration
 - transformation matrix (table), 121
- construction lines
 - modifying, 69
 - rays, 69
 - xlines, 69
- ContourlinesPerSurface property, 255
- Contrast property, 287
- control codes
 - text formatting (table), 169
- ControlPoints property
 - splines, 127
- controls for forms
 - adding code, 315
 - adding to forms, 314
 - control toolbox, 314
 - for formatting in VBA, 315
 - modifying, 314
- conversion functions
 - operation methods, 40
- converting coordinates, 249
- coordinates
 - converting between systems, 249
- Coordinates property
 - polylines, 126
- Copy method, 109, 163, 181, 297
- copying
 - arraying, 108
 - mirroring, 108
 - multiple objects, 109
 - offsetting, 108
 - single object, 109
- CopyObjects method, 109
 - example code, 109
- Count property, 38
 - for menus, 204
- CreatePoint object
 - example code, 88
- CreateRegion
 - example code, 90
- CreateSolid object
 - example code, 88
- CreateSpline object
 - example code, 86
- CreateTypedArray method, 47, 73
- creating lines
 - Add Polyline, 85
 - AddLightweightPolyline method, 85
 - AddLine method, 85
 - AddMLine method, 85
- creating objects
 - graphical, 38
 - in blocks, 84
 - in model space, 84
 - in paper space, 84
 - nongraphical, 38
- crosshairs
 - resizing, 56
- cursor
 - restricting with Ortho mode, 69
 - restricting with Snap mode, 68
- cursor menus
 - adding new items, 228
 - creating, 202, 203
 - deleting, 203
 - on graphics screen, 202
 - Shortcut menu property, 228
 - truncating, 202

- CursorSize property
 - crosshairs, resizing, 56
- curved objects
 - arcs, 86
 - circles, 86
 - ellipses, 86
 - spline curves, 86
- CustomDictionary property, 170
- cylinder, example code for adding, 260

D

- DatabasePreferences object
 - storing options, 56
- DCS
 - converting coordinates, 250
 - definition, 250
- DDIM Annotation dialog box, 183
- DDIM Format dialog box, 183
- DDIM Geometry dialog box, 183
- Deactivate event, 239
- Declare statement, 332
- Default to Full Module View
 - Project window, 28
- default VBA project name, 15
- Degree property
 - splines, 128
- Delete method, 62
 - collections, 119
 - deleting views, example code, 62
 - example code, 209
 - in selection sets, 107
 - layers, 139
 - linetypes, 142
- DeleteFitPoint method
 - splines, 128
- deleting
 - collection member, 44
- demand loading
 - and indexes, 308
 - performance benefits, 308
- Description property
 - example code, 142
 - linetypes, 142
- design mode, VBA environment and, 313
- Detach method
 - example code, 304
 - external references, 304
- dialog boxes
 - in modal forms, 317
 - macro, 16
 - options for VBA IDE, 28
 - Save Project, 26
 - VBA Manager, 13
- dialog controls. *See* controls
- DIESEL string expressions, 204
- DimAligned object
 - example code, 182, 188
- DimAngular object
 - example code, 179
- DIMASO system variable, 175
- DIMCLRD system variable, 194
- DIMCLRT system variable, 192, 194
- dimension lines
 - illustration, 173
- dimension styles
 - active, 175
 - Add method, 183
 - creating dimensions, 182
 - modifying, 183
 - parent, 183
- dimension styles (illustration), 182
- dimension system variables
 - list, 173, 177
- dimensions
 - and geometry, 189
 - angular, 172
 - angular (illustration), 178
 - annotations, 174
 - associative, 175
 - hook lines (illustration), 177, 190
 - in model space, 189
 - in paper space, 189
 - leader lines (illustration), 174, 190
 - LeaderLength setting, 178
 - linear, 172
 - methods, editing, 181
 - modifying, 175
 - ordinate (illustration), 179
 - properties, editing, 181
 - radial, 172
 - creating, 177
 - rotating, 176
 - text styles, 174
 - illustration, 174
 - types of illustration, 172
- DIMFIT system variable, 177
- DIMGAP system variable, 194
- DIMJUST system variable, 173, 177
- DIMLFAC system variable, 189
- DimOrdinate object
 - example code, 180
- DimRadial object
 - example code, 178
- DIMTAD system variable, 173, 177
- DIMTIH system variable, 173, 177
- DIMTOFL system variable, 173, 177
- DIMTOH system variable, 173, 177
- DIMTXSTY system variable, 192
- DIMTXT system variable, 192, 194
- DIMTXTSTY system variable, 194
- Direction property, 271
 - example code, 252, 271

- DirectionVector property, 70
 - example code, 70
- displacement vector, 116
- Display Coordinate System
 - definition, 250
- Display method
 - in viewports, 269
- Display preference, Options dialog box, 55
- Display property
 - example code, 270, 271
- displaying
 - macros, 17
- DisplayPlotPreview method, 277
- DisplayScreenMenu property, 56
- DisplayScrollBar property, 56
- DistanceToReal method, 73
- distributing applications
 - VB, 324
 - VBA, 324
- Dock method
 - docking toolbars, 219
 - example code, 219
- document level events
 - and Document object, 240
 - coding, 241
 - declaring objects with events, 240
- document- level events
 - enabling, 237
- Document object
 - ActiveViewport property, 64
 - as AutoCAD drawing, 36
 - declaring objects with events, 240
 - enabling events, 237
 - events (list), 237
 - example code, 241, 270, 271
 - model space, accessing, 36
 - modifying drawings, 54
 - paper space, accessing, 36
 - ZoomAll method, 62
 - ZoomExtents method, 62
- Document window
 - creating views, 62
 - Delete method, 62
 - displaying views, 58
 - maximizing, 58
 - minimizing, 58
 - modifying position, 58
 - Regen method, 66
 - sizing, 58
 - Split method, 64
 - status of active document, 58
 - Update method, 66
 - viewports, 63
 - WindowState property, 58
 - Zoom option, 59
 - zoom scale, specifying, 60
 - ZoomAll method, 61
- Document window (*continued*)
 - ZoomCenter method, 61
 - ZoomExtents method, 61
 - ZoomPickWindow, 59
 - ZoomWindow, 59
- document-level events
 - events (list), 237
 - right-clicking, 237
- Documents collection
 - links to drawing, 36
- Drafting preference, Options dialog box, 55
- Drag and Drop Text Editing
 - Project window, 28
- drawing display, defined, 34
- drawing extents
 - and rays, 71
 - zooming to, 61
- drawings
 - performing calculations, 67
 - text styles, 153
- drawing-stored options, setting, 56
- DWF files
 - exporting to, 80
- DXF codes, and filter types (table), 98
- DXF files
 - exporting to, 80
 - importing, 80

E

- editing
 - 2D objects, 107
 - macros, 18
 - nongraphical objects, 107
- ElevateOrder method
 - splines, 128
- Ellipse object
 - creating, 86
- embedded projects
 - auto embed feature, 19
 - ThisDrawing object, 40
- enabling, application-level events, 235
- EndAngle property, 124
- EndCommand event, 232, 234, 239
- EndLISP event, 234, 239
- EndModal event, 235
- EndOpen event, 232, 235
- EndPlot event, 235, 239
- EndPoint property, 124
 - example code, 124
- EndSave event, 235, 239
- EndShortcutMenu event, 239
 - example code, 241
- EndTangent property
 - splines, 127
- EPS (Encapsulated PostScript) files
 - exporting to, 80

- ERASE command
 - macros, 227
- Erase method, 112, 163, 181, 297
 - in selection sets, 107
 - leader lines, 191
- Err object, 321
- errors
 - at runtime, 318
 - Break mode, 19
 - at compile time, 318
 - Err object, 321
 - error messages, 318
 - error trapping, 318
 - ignoring errors, 319
 - user input control, 321
- Evaluate method, 94
 - example code, 94, 130
 - in hatches, 129
 - leader lines, 192
- event handlers
 - and infinite loops, 233
 - and interactive functions, 233
 - and parameters, 232
 - and subroutines, 232
 - Begin events, 232
 - creating objects, 233
 - deleting objects, 233
 - End events, 232
 - writing guidelines, 232
- events
 - document level, coding, 241
 - enabling application-level events, 235
 - enabling object-level events, 242
 - handling document-level events, 237
 - in AutoCAD, 232
 - InitializeEvents procedure, 241
- Excel, extracting attributes, example code, 330
- Explode method, 291
 - example code, 292
- exploding
 - blocks, 124
 - objects, 124
 - polylines, 124
- Export method, 80
 - for saved layer settings, 151
- extended data
 - uses of, 38
 - with graphical objects, 38
- external references
 - attaching, 302
 - binding, 306
 - defined, 301
 - demand loading, 308
 - detaching, 304
 - and indexes, 308
 - overlying, 303
 - reloading, 305

- external references (*continued*)
 - unloading, 305
 - updating, 302
- ExternalReference object
 - example code, 302, 304, 305, 306, 307

F

- Fade property, 287
- FieldLength property, 296
- Files preference, Options dialog box, 55
- FILLMODE system variable, 88
- filter lists, 97–104
 - example code, 98–104
- filter types, and DXF codes (table), 98
- filtering
 - example code, 98
 - selection sets, 97–104
- FitPoints property
 - splines, 127
- FitTolerance property
 - splines, 127
- Float method
 - floating toolbars, 218
- flyout button, 214
- Flyout property
 - ToolbarItem object, 221
- flyout toolbars
 - AddToolBarButton method, 217
 - example code, 217
- FontFile property, 154, 156
 - example code, 156
- FontFileMap property, 169
- fonts
 - alternative, specifying, 170
 - and text styles, 169
 - assigning in drawings, 154
 - Big Font files, 156
 - exporting in drawings, 155
 - font mapping tables, 169
 - SHX fonts, 155
 - specifying, 165
 - substituting, 169
 - substitution rules, 170
 - TEXTFILL system variable, 155
 - TrueType, 155
 - Unicode, 156
- form controls. *See* controls
- Format menu
 - VBA IDE, 315
- formatting characters
 - example code, 167
 - MText object, 165
- forms
 - adding controls, 314
 - control toolbox, 314
 - displaying, 316

- forms (*continued*)
 - formatting controls in VBA, 315
 - hiding, 316
 - importing into VBA, 22
 - loading at runtime, 316
 - as macros, 316
 - modal dialog boxes, 317
 - modal versus modeless, 313
 - modal, defined, 316
 - mode, setting, 313
 - modifying controls, 314
 - unloading, 316
 - VBA environment, 313
 - VBA project component, 21
 - visibility at runtime, 315
- Freeze property, 137

G

- geometric tolerances
 - creating, 193
 - modifying, 193
- GetAttributes method, 299
 - example code, 299, 330
- GetBitmaps method, 215
 - example code, 216
- GetDistance method, 73
 - example code, 73
- GetFitPoint method
 - splines, 128
- GetFont method
 - example code, 155
- GetKeyword method, 76
 - example code, 76
- GetPoint method, 76
 - defined, 76
 - example code, 76, 248
- GetString method, 76
 - example code, 75
- GetUCSMatrix method, 247
- GetVariable method, 67
- GetXData method
 - example code, 310
- global projects
 - defined, 12
 - ThisDrawing object, 40
- graphical objects
 - defined, 34
 - editing commands, 38
 - properties, 38
 - visible objects, 38
- GridOn property, 271

H

- Hatch object
 - AppendInnerLoop method, 93
 - AppendOuterLoop method, 93
 - AssociativeHatch property, 129
 - associativity, 92, 129
 - boundaries, 93
 - Boundary Hatch dialog box, 131
 - creating, 92
 - editing, 129, 132
 - example code, 94, 130, 132
 - handling islands, 94
 - patterns, 92, 131
 - specifying loops, 92
- hatches, editing, 129
- HatchStyle property
 - definitions (table), 94
 - handling islands, 94
- Height property, 154, 157, 286, 297
 - example code, 157, 272
- Help, status line
 - for menus and toolbars, 227
- HelpString property
 - ToolBarItem object, 214, 220
- hierarchy
 - accessing objects in VBA, 40
 - referencing objects, 40
- hook lines
 - illustration, 190

I

- image boundaries, defined, 284
- ImageFile property, 284
- ImageHeight property, 286
- ImageVisibility property, 286
- ImageWidth property, 286
- Import method
 - file conversions, 80
 - for saved layer settings, 151
- importing
 - forms, 22
 - modules, 22
- Index property
 - ToolBarItem object, 221
- INDEXCTL system variable, 308
- indexes
 - and demand loading, 308
 - and external references, 308
- InitializeUserInput
 - example code, 77
- InitializeUserInput method, 75, 321
 - (list), 77
 - defining keywords, 77
- InsertBlock method, 290
 - example code, 291, 292

- InsertInMenuBar method, 198
 - example code, 202, 205–208, 212, 228
 - from PopupMenu object, 200
- InsertionPoint property, 160, 297
- InsertLoopAt method, 130
- InsertMenuInMenuBar method, 198
 - example code, 200
 - from PopupMenus collection, 200
- intersecting regions
 - Region object, 91
- IsPeriodic property
 - splines, 128
- IsPlanar property
 - splines, 128
- IsRational property
 - splines, 128
- Item method, 38
 - iterating collections, 43
- iterating collections
 - example code, 133

J

- justification, format codes, 166

K

- keywords
 - command line, 77
 - user input keywords, 77
- Knots property
 - splines, 127

L

- Label property
 - accelerator key, 206
- Labels for menus
 - captions, 205
 - special codes, 205
- LAYER command
 - macros, 225
- Layer object
 - example code, 108, 134, 136
- layer properties, saving. *See* layer settings, saving
- Layer property, 144, 271
 - raster images, 285
- layer settings
 - deleting saved settings
 - example code, 150
 - exporting saved settings, 151
 - example code, 152
 - importing saved settings, 151
 - example code, 152
 - listing saved settings, 148
 - managing, 148

- layer settings (*continued*)
 - renaming saved settings
 - example code, 150
 - restoring saved settings, 151
 - example code, 151
 - saving, 147, 149
 - example code, 150
 - illustrated, 147
 - storing, 146–150
- LayerOn property, 136
 - example code, 136
- layers, 133
 - ACI number, 138, 139
 - ActiveLayer property, 136
 - and color, 133
 - assigning linetypes, 139
 - assigning to objects, 143
 - and blocks, 139
 - changing layers, 144
 - Color property, 138
 - color, assigning, 138, 139
 - freezing, 137
 - locking, 138
 - plotting, 136
 - setting color, 144
 - standard colors (table), 140
 - turning off (illustration), 136
- Layers collection, 133
 - example code, 133
 - saved layer settings and, 147
- LayerStateManager object, 147, 148
 - accessing, 149
 - associating database with, 149
- Layout object, 265
 - example code, 278
- layouts
 - Block object, 265
 - Block property, 265
 - CanonicalMediaName property, 265
 - Layout object, 265
 - lineweight scale, 267
 - in model space, 264
 - paper size, 265
 - in paper space, 264
 - PaperUnits property, 265
 - plot elements, 265
 - PlotConfiguration object, 265
 - plotting input values (list), 266
 - switching model space and paper space, 270
- LayoutSwitched event, 239
- leader lines
 - annotations, 191
 - associativity and editing, 192
 - associativity with annotations, 191
 - color, 190
 - creating, 190

- leader lines (*continued*)
 - determining type, 190
 - illustration, 189
 - modifying, 189
 - scale, 190
 - scaling, 192
 - updating geometry, 192
- Leader object
 - example code, 190, 191
- LensLength property, 271
- LightweightPolyline object, 126
 - creating, 85
 - example code, 85, 111, 112
- LIN library files
 - and linetypes, 140
- Line object
 - creating, 85
 - example code, 122, 124
- linear dimensions
 - aligning, 176
 - creating, 176
 - illustration, 172
 - modifying properties (illustration), 176
 - rotating, 176
- lines
 - creating, 85
 - lengthening, 124
 - Ray object, 69
 - Xline object, 69
- Linetype object
 - example code, 141, 142
- Linetype property, 139, 271
 - defining for layers, 145
 - example code, 145
 - raster images, 285
- layers
 - and linetypes, 133
- linetypes
 - active, 141
 - assigning to layers, 139
 - assigning to objects, 143
 - changing descriptions, 142
 - complex, 140
 - continuous (illustration), 145
 - deleting, 142
 - examples of (illustration), 140
 - LIN library files, 140
 - Load method, 140
 - new object properties, 141
 - renaming, 141
 - scales (illustration), 142
 - x-ref dependent, 141
- Linetypes collection, 133
- LinetypeScale property, 142, 271
- lineweights
 - scaling in layouts, 267
- LISPCancelled event, 235, 239

- listing
 - ObjectARX applications, 36
- Load method
 - example code, 141, 198
 - linetypes, 140, 145
 - MenuGroups collection, 198
- loading
 - ObjectARX applications, 36
 - VBA projects on startup, 322
- loading projects
 - in AutoCAD, 13
 - in VBA, 27
- Lock property, 138
- loops
 - defining regions, 89
- LowerLeftCorner property
 - illustration, 64
- LTSCALE system variable, 142

M

- macro libraries, 26
- Macro property, 205, 322
 - ToolStripItem object, 214, 220
- macros
 - and projects, 12
 - backslash character, 225
 - canceled commands, 226
 - command handling, 226
 - delays (list), 225
 - displaying, 17
 - editing, 18
 - enabling virus protection, 19
 - ERASE command, 227
 - guidelines for writing, 222
 - in menus, 222
 - in toolbars, 222
 - in VBA, 19
 - LAYER command, 225
 - naming, 17
 - object selection, single, 227
 - running, 18
 - running on startup, 322
 - SELECT command, 225
 - setting Break mode, 19
 - special characters (table), 222
 - stepping into, 18
 - terminating, 224
 - use of repetition, 226
 - user input, 225
- Macros dialog box, 16
 - VBARUN command, 31
- MainDictionary property, 170
- matrix, configuring transformation matrix
 - (table), 121
- maximizing, Document window, 58
- MAXSORT system variable, 67

- MClose property, 252
- menu macros
 - special characters (table), 222
- menu objects
 - ActiveX (illustration), 197
 - See also* PopupMenu objects
- MenuBar collection, 196
 - inserting menus, 200
 - object model (illustration), 198
 - rearranging menus, 202
 - removing menus, 201
- MenuBar object
 - example code, 202
- MenuBar property
 - example code, 202
- MenuGroup object
 - example code, 198, 199, 200
- MenuGroups collection, 196
 - Load method, 198
 - modifying (list), 199
 - new groups, creating, 199
 - object model (illustration), 196
 - and popup menus, 197
 - and toolbars, 197
- menus
 - assigning menu items, 211
 - checking, 211
 - creating submenus, 207
 - deleting, 209
 - disabling, 211
 - enabling, 211
 - positioning, 211
 - returning submenus, 211
 - type of menu item, 211
- meshes
 - density, defined, 252
 - polyface mesh, creating, 253
 - rectangular, defined, 252
 - and solids, 252
 - and wireframes, 252
- methods. *See specific method names*
- minimizing, Document window, 58
- MInsertBlock object, 290
- Mirror method, 112, 163, 181
 - example code, 112
 - illustration, 112
- Mirror3D method, 258
 - example code, 259
- mirroring
 - example code, 112
 - in 3D, 258
 - objects, 108
 - Text objects, 112
 - with two coordinates, 112
- MIRRTTEXT system variable, 112, 163
- MLine object
 - creating, 85

- Mode property, 297
- model space
 - accessing objects, 36
 - and paper space, switching, 270
 - creating objects, 84
 - defined, 264
 - dimensioning in, 189
 - example code, 84
 - hidden lines, 276
 - layouts, 264
 - plot settings, modifying, 278
 - plotting, 276
 - referencing in ActiveX code, 84
 - returning objects, 41
 - viewports, 267
- ModelSpace property, 84
- Modified event, 242
- module
 - VBA object component, 21
- modules
 - importing into VBA, 22
- MomentOfInertia property, 255
- Move method, 163, 181, 192
 - vectors, 116
- Move object
 - example code, 117
 - illustration, 116
- MSpace property, 269
 - example code, 270, 271
- mtext
 - control codes (table), 169
 - creating, 163
 - in drawings, 153
 - modifying, 163
 - Unicode fonts (table), 168
 - uses for, 163
- MText object, 163
 - creating text, example code, 167
 - example code, 164, 191
 - formatting codes (table), 165
 - formatting options, 164
 - justification (illustration), 164
 - leader lines, 191
 - modifying, 164
 - orientation options, 164
 - overriding default, 164
 - text boundary, 164
- multiline text. *See* mtext

N

- Name property
 - example code, 108
 - layers, 134
 - linetypes, 141
 - PopupMenu collection, 203
 - raster images, 284

- Name property (*continued*)
 - setting active viewport, 64
 - ToolStripItem object, 214, 220
 - toolbars, 212
- named objects
 - character length, 108
 - purging, 107
 - renaming, 108
 - specifying, 107
- naming macros, 17
- naming projects, in VBA IDE, 25
- NClose property, 252
- NewDrawing event, 235
- nonbreaking spaces, inserting, 165
- nongraphical objects, editing, 107
- NumberOfControlPoints property
 - splines, 128
- NumberOfCopies property, 277
 - example code, 278, 279
- NumberOfFitPoints property
 - splines, 128
- NURBS, AddSpline method, 86

O

- Object Coordinate System
 - definition, 249
- object level events
 - and event handlers, 243
 - and VBA keywords, 242
 - class modules, 242
 - enabling, 242
- object libraries
 - AutoCAD as Automation controller, 3
 - referencing in VBA IDE, 327
- object methods, defined, 45
- object model
 - MenuBar collection (illustration), 198
 - MenuGroups collection (illustration), 196
- ObjectAdded event, 239
- ObjectARX applications
 - specific actions, 36
- ObjectErased event, 239
- ObjectModified event, 239
- object-oriented, VBA interface, 3
- objects
 - active, resetting, 67
 - changing layers, 144
 - closed
 - calculating area (illustration), 74
 - defining from user input points, 74
 - creating, 84
 - enabling events, 242
 - existing, modifying, 107
 - exploding, 124
 - extending, 124
 - for menus, 196

- objects (*continued*)
 - graphical, 38
 - in ActiveX, 34
 - in ActiveX (list), 3
 - moving along a vector, 116
 - named, specifying, 107
 - nongraphical, 38
 - object component, 21
 - offsetting, 110
 - open, calculating area (illustration), 74
 - properties, defined, 45
 - removing from selection sets, 106
 - rotating in 3D, 256
 - scale factor, 119
 - scaling, 119
 - ThisDrawing, 18
 - transforming, 121
 - trimming, 124
 - See also specific object names*
- ObliqueAngle property, 154, 157, 160
- obliquing angles, in text
 - illustration, 157
 - setting, 166
- OCS
 - converting coordinates, 249
 - creating polylines, 85
 - definition, 249
- Offset method, 110
 - creating objects, 45
 - example code, 111
- offsetting, objects, 108
- On Error
 - forms of (list), 319
 - VBA statement, example code, 320
- Open method
 - Documents collection, 54
 - example code, 54
- Open VBA Project dialog box, 323
- opening VBA IDE
 - from command line, 20
 - from menu bar, 20
- OpenSave preference, Options dialog box, 55
- operating system
 - rebooting, 5
 - requirements for VBA, 5
- options
 - setting for VBA projects, 19
- Options dialog box
 - in VBA IDE, 28
 - preferences objects (list), 55
- ordinate dimensions
 - and error preventing, 179
 - creating, 179
 - leader lines, 179
- organizational structures, defined, 34
- organizing projects, with VBA Manager, 13
- Origin property, 286

- Ortho mode
 - cursor movement, 69
 - defining axes (illustration), 69
 - example code, 69
- OrthoOn property, 69
- Output preference, Options dialog box, 55
- overlaying, external references, 303

P

- paper space
 - accessing objects, 36
 - and model space, switching, 270
 - creating objects, 84, 274
 - defined, 264
 - dimensioning in, 189
 - editing models, 267
 - example code, 84
 - hidden lines, 276
 - layouts, 264
 - plotting, 276
 - referencing in ActiveX code, 84
 - returning objects, 41
 - scaling linetypes, 275
 - scaling views, 274
 - viewports, floating, 267
 - zooming (illustration), 275
- Paper Space Display Coordinate System
 - definition, 250
- PaperSpace property, 84
- PaperUnits property, 265
- parent objects
 - relation to root object, 45
- Parent property
 - ToolStripItem object, 221
- PatternAngle property, 132
- PatternDouble property, 132
- PatternName property, 132
- patterns
 - assigning hatch patterns, 93
- PatternScale property, 132
- PatternSpace property, 132
 - example code, 132
- PatternType constants
 - defined, 93
- pausing macros
 - delays (list), 225
- PDMODE system variable
 - illustration, 87
- PDSIZE system variable
 - illustration, 87
- PICKADD system variable, 222
- PICKAUTO system variable, 222
- Plot object, 276
 - example code, 278, 279
 - Plot dialog box, 40
- PlotConfiguration object, 265
- PlotHidden property, 276
- plotting
 - in model space, 276, 278
 - in paper space, 276, 279
 - layout input values (list), 266
 - methods (list), 276
 - properties (list), 276
- PlotToDevice method, 277
 - example code, 278, 279
- PlotToFile method, 277
- PlotType property
 - example code, 278
- Point object
 - controlling appearance of, 87
 - creating, 87
 - example code, 88
- polar arrays
 - center point, specifying, 113
 - creating, 113
 - example code, 114
 - reference points, 114
- PolarPoint method, 73
- PolyfaceMesh object
 - example code, 254
- PolygonMesh object, 252
- Polyline object, 126
 - creating, 85
 - defining from user input points, 74
- polylines
 - creating in OCS, 85
 - editing, 126
 - exploding, 124
 - fit and spline fit, 126
 - modifying, 126
- PopupMenu object
 - creating submenus, 207
 - deleting menu items, 209
 - example code, 200, 202, 203
 - InsertInMenuBar method, 200
 - menu items, ordering (illustration), 204
- PopupMenuItem object
 - accelerator keys, assigning, 206
 - AddSeparator method, 206
 - Caption property, 210
 - Check property, 210
 - creating, 204
 - deleting menu items, 209
 - editing labels, 205
 - Enable property, 210
 - example code, 200
 - HelpString property, 210
 - Index property, 211
 - index, defined, 204
 - Label property, 209
 - label, defined, 204
 - Macro property, 210
 - macro, defined, 205

- PopupMenu object (*continued*)
 - new menu items, adding, 204
 - Parent property, 211
 - position menu, changing, 204
 - status-line Help message, 227
 - Submenu property, 211
 - Tag property, 209
 - tag, defined, 205
 - Type property, 211
 - writing macros, 222
- PopupMenu collection
 - cursor menus, 203
 - InsertMenuInMenuBar method, 200
 - Name property, 203
 - new menus, creating, 203
 - RemoveMenuFromMenuBar method, 201
- PostScript files, importing, 80
- pound sign (#)
 - stacked text indicator, 165
- preferences in AutoCAD
 - accessing, 56
 - CursorSize property, 56
 - drawing-stored options, 56
 - properties, 56
 - registry-stored options, 56
- Preferences object, Options dialog box, 39
- Preferences objects, accessing, 56
- PrincipalDirections property, 255
- PrincipalMoments property, 255
- Procedure Separator Display
 - Project window, 28
- ProductOfInertia property, 255
- Profiles preference, Options dialog box, 55
- project components
 - forms, 21
 - importing, 22
 - modules, 21
 - new, 21
 - references, 21
- Project window
 - defined in VBA IDE, 20
 - settings, 28
- projects
 - .dwb file, 25
 - circular references, 27
 - components in VBA, 21
 - embedded, 12, 14, 30
 - extracting, 15
 - global, 12, 30
 - importing, 22
 - loading in AutoCAD, 13
 - naming in VBA IDE, 25
 - referencing, 26
 - setting options for VBA, 19
 - unloading, 14
 - VBA default name, 15
 - VBA terms defined, 30
- PromptString property, 297
- Properties window
 - VBA IDE, 25
- properties. *See specific property names*
- PSDCS
 - converting coordinates, 250
 - definition, 250
- PSLTSCALE system variable, 275
- pull-down menus
 - creating, 202
 - from menu bar, 202
 - truncating, 202
- PurgeAll method
 - example code, 107
- purging, named objects, 107
- PViewport object, 268
 - example code, 270, 271, 272

Q

- QuietErrorMode property, 277
- Quit method
 - example code, 330

R

- radial dimensions
 - creating, 177
 - illustration, 172
 - TextPosition property, 177
- RadiiOfGyration property, 255
- raster images
 - and vector files, 282
 - attachments, 282
 - bitonal images, 286
 - clipping (illustration), 287
 - clipping boundaries, 287
 - display properties (list), 287
 - display, adjusting, 287
 - DPI, defined, 282
 - file naming, 284
 - geometry scale, 282
 - hiding boundaries, 285
 - image boundaries, defined, 284
 - image names vs. file names, 284
 - linking paths, 282
 - modifying (list), 284, 286
 - Name property, 284
 - path, modifying, 284
 - properties for modifying (list), 285
 - redraw speed, 286
 - resolution, 282
 - scale factor, 282
 - showing boundaries, 285
 - vectors, 287
 - visibility, 286

- Raster object, 282
 - example code, 283, 288
- Ray object, 69, 71
 - BasePoint property, 71
 - DirectionVector property, 71
 - example code, 71
- rectangular arrays, 115
 - creating in 3D, 258
 - example code, 115
 - in 3D (illustration), 258
 - snap rotation angle, 115
- reference points, in polar arrays, 114
- referenced projects, loading in AutoCAD, 27
- referencing objects
 - calling hierarchy, 40
 - type library, 45
- Regen method
 - example code, 272
 - for text styles, 154
- Region object
 - Boolean method, 90
 - calculating total number, 89
 - creating composite, 90
 - defining loops, 89
 - example code, 90
 - intersecting regions, 91
 - subtracting, 90
 - uniting regions, 91
- registry-stored settings, 55
 - accessing, 36
- Reload method
 - example code, 305
 - external references, 305
- RemoveFromMenuBar method
 - example code, 202
- RemoveHiddenLines property, 276
- RemoveItems method
 - in selection sets, 106
- RemoveMenuFromMenuBar method
 - from PopupMenus collection, 201
- RenderSmoothness property, 255
- Require Variable Declaration
 - Code window, 28
- Reverse method
 - splines, 128
- right-click menu. *See* cursor menu
- root object
 - accessing, 41
 - hierarchy, 41
 - relation to parent object, 45
 - See also* Application object
- Rotate method, 117, 163, 181
 - example code, 118
- Rotate3D method, 256
 - example code, 257
- rotating objects, 117
 - illustration, 117

- rotation angles, 157
- Rotation property, 160, 164, 181, 297
- run mode, in VBA environment, 313
- running macros, 18
- runtime
 - error trapping, 319
 - errors in VBA, 318
 - object referencing, 46

S

- sample code, finding for ActiveX and VBA, 8
- SAT files
 - exporting to, 80
 - importing, 80
- SAVE command
 - from VBA IDE, 26
- Save method
 - for layer settings, 149
 - verifying changes, 55
- Save Project dialog box
 - accessing, 26
 - VBAUNLOAD command, 32
- SaveAs method, 54
 - example code, 199
- saved layer settings. *See* layer settings
- saving layer settings. *See* layer settings, saving
- saving projects, in VBA IDE, 26
- scale factor
 - illustration, 119
 - object dimensions, 119
- ScaleEntity method, 119, 181, 192, 286, 297
 - example code, 120
- ScaleFactor property, 161, 297
- scaling
 - in layouts, 266
 - in paper space, 274
 - leader lines, 192
 - linetypes, 275
 - objects, 119
 - viewports (illustration), 274
- SELECT command
 - macros, 225
- Select method, 96
- SelectAtPoint method, 96
- SelectByPolygon method, 96
- Selection preference, Options dialog box, 55
- selection sets
 - filter lists, 97–104
 - removing objects, 106
- SelectionChanged event, 239
- SelectionSet object, 95
 - example code, 96, 97, 98, 309, 310
- SelectionSets collection, 95
- SelectOnScreen method, 96
 - example code, 97, 309

- separators
 - adding to menus, 206
 - adding to toolbars, 215
- SetBitmaps method, 215
 - LargeIconName parameter, 216
 - SmallIconName parameter, 215
- SetBulge method
 - polylines, 126
- SetControlPoint method
 - example code, 129
 - splines, 128
- SetCustomScale method, 266
- SetDatabase method, 149
- SetFitPoint method
 - splines, 128
- SetFont method
 - example code, 155
- SetLayoutsToPlot method, 277
 - example code, 279
- SetPattern method, 132
- SetProjectFilePath method, 284
- setting crosshairs
 - example code, 56
- SetVariable method, 67, 87
 - example code, 161
 - for dimensioning, 173
 - linetypes, 142
- SetWeight method
 - splines, 128
- SetWidth method
 - polylines, 126
- SetXData method
 - example code, 309
- sharing code by referencing projects in VBA, 26
- ShortcutMenu property, 228
- ShowRotation method, 286
- SHX fonts, 155
- Single object selection in macros, 227
- sizing
 - Application window, 57
 - Document window, 58
- slash (/)
 - stacked text indicator, 165
- SliceSolid method
 - example code, 262
- snap angle
 - illustration, 68
- Snap mode
 - cursor movement, 68
- snap rotation angle
 - rectangular arrays, 115
- SnapBasePoint property, 68
 - example code, 68
- SnapRotationAngle property, 68, 115
 - example code, 68
- Solid object
 - analyzing properties, 255
 - Boolean intersection (illustration), 260
 - CheckInterference method, 260
 - combining solids, 260
 - creating, 88
 - creating solids, 255
 - example code, 88
 - properties (list), 255
- solid-filled areas
 - illustration, 88
 - See also* Solid object
- spaces
 - inserting nonbreaking spaces, 165
- spell checking, 170
- Spline object
 - creating, 86
 - example code, 86, 129
- splines
 - editing, 127
 - querying, 128
- SPLINETYPE system variable, 126
- split bar, in Code window, 24
- Split method, 64
 - viewports, example code, 64
- splitting viewports
 - example code, 65
- spreadsheet
 - extracting attributes to a spreadsheet, example code, 330
- stacked text, creating, 165
- StandardScale property, 266
 - example code, 278
- StartAngle property, 124
- StartBatchMode method, 277
- StartPoint property, 124
- StartTangent property
 - splines, 127
- startup
 - loading VBA projects, 322
 - running a macro, 322
- status-line Help, for menus and toolbars, 227
- stepping into macros, 18
- style settings, 34
- StyleName property, 160, 164, 181
- submenus
 - adding, 207
 - populating, 208
 - positioning, 207
- subtracting regions
 - Boolean method, 90
 - Region object, 91
- System preference, Options dialog box, 55
- system requirements, for AutoCAD VBA, 5
- SysVarChanged event, 235

T

- Tag property, 205
 - ToolStripItem object, 220
- TagString property, 297
 - attribute references, 299
 - example code, 299
- terminating macros
 - code examples, 224
- terminology in VBA environment (list), 312
- text
 - obliquing angle, 166
 - overlining, 165
- text height, format codes, 165
- Text object
 - aligning in drawings (illustration), 161
 - angles, setting, 157
 - display backward, 158
 - display upside down, 158
 - example code, 157, 159, 161
 - formatting, 160
 - height settings, 156
 - line text, 153
 - line text, creating, 159
 - methods (list), 163
 - mirroring text, 112
 - modifying, 163
 - mtext, 153
 - ObliqueAngle property, 157
 - properties (list), 160
 - spell check, 170
 - text generation flag, 158
 - used in drawings, 153
- text styles
 - changing properties, 154
 - creating, 153
 - current, 153
 - default, 153
 - for dimensions, 174
 - properties (table), 153
- TextAlignmentPoint property, 161
 - example code, 161
- TEXTFILL system variable, 155
- TextGenerationFlag property, 154, 158, 161
 - example code, 158, 159
- TextOverride property
 - example code, 182
- TextPosition property, 181
- TextRotation property, 181
- TextString property, 161
 - attribute references, 299
 - example code, 299
- TextStyle object, 153
 - example code, 156
 - properties (list), 154
- TextStyles collection, 153
- TextSuffix property
 - example code, 188
- ThisDrawing object, 31
 - accessing Document object, 40
 - in embedded objects, 18, 40
 - in global objects, 18
 - in global projects, 40
 - working in zero document state, 323
- TILEMODE system variable, 269
- Tolerance object
 - example code, 193
 - leader lines, 191
- tolerances
 - geometric, 193
 - system variables (list), 194
- toolbar macros
 - special characters (table), 222
- Toolbar object
 - AddSeparator method, 215
 - AddToolStripButton method, 213
 - Dock method, 219
 - Docked property, 219
 - example code, 213, 214, 217, 219
 - Float method, 218
 - flyout toolbars, creating, 217
 - naming, 213
 - using Type property, 215
- ToolStripItem object, 213
 - creating flyout button, 214
 - deleting, 220
 - example code, 214, 216, 217
 - Flyout property, 221
 - GetBitmaps, 215
 - HelpString property, 214, 220
 - Index property, 221
 - Macro property, 214, 220
 - Name property, 214, 220
 - Parent property, 221
 - positioning toolbar buttons, 213
 - SetBitmaps method, 215
 - status-line Help message, 227
 - Tag property, 220
 - Type property, 221
 - writing macros, 222
- toolbars
 - modifying, 199
- Toolbars collection
 - Add method, 212
 - Name property, 212
- transformation matrix
 - assigning matrix to variable, 121
 - rotation (table), 122
 - scaling (table), 122
 - translation (table), 122
 - user coordinate system, 247
 - world coordinate system, 247

- TransformBy method, 121
 - example code, 122
- TranslateCoordinates method, 73
 - converting coordinates, 249
 - example code, 248, 250
- TrueType fonts, 155
 - height settings, 156
- type library
 - adding reference, 46
 - defined, 45
- Type property
 - example code, 216
 - ToolStripItem object, 221
- TypeName function, 48

U

UCS

- converting coordinates, 249
- definition, 249

UCSIconAtOrigin property, 247

UCSIconOn property, 247

UCSORG system variable, 247

Unicode fonts, 156

- table, 168

uniting regions

- Region object, 91

Unload method

- example code, 306
- external references, 305

unloading

- ObjectARX applications, 36

Update method, 124

- example code, 66
- for text styles, 154
- redrawing objects, 107
- Text object, 161

UpperRightCorner property

- illustration, 64

User Coordinate System

- axis location, 247
- definition, 249
- origin point location, 247
- viewports, 247

user input functions

- prompting, 40

user input methods

- GetInteger method, 75
- GetKeyword, 76
- GetPoint, 76
- GetPoint method, 75
- GetString, 76
- GetString method, 75

user input pausing macros, 225

User preference, Options dialog box, 55

UserForm window

- editing forms, 23

UseStandardScale property, 266

Utility object

- calculating area, 73
- calculation methods (list), 72
- example code, 248
- functions, 40
- TranslateCoordinates method, example code, 250
- user input methods, 75

V

variants

- CreateTypedArray method, 47
 - defined, 46
 - input arrays, 47
 - output arrays, 47
 - polyline editing, 126
 - Typename function, 48
 - VarType function, 48

VarType function, 48

VBA environment

- accessing applications, 327
- application distribution, 324
- control toolbox (illustration), 312
- design mode, 313
- dialog boxes, creating, 312
- forms, 312, 313
- password protection, 321
- referencing object libraries, 327
- run mode, 313
- running macros, 322
- terminology (list), 312
- Windows API procedures, accessing, 332
- working in a zero document state, 323

VBA Hide method, 316

VBA IDE

- "Hello World" exercise, 29
- defined, 31
- macro execution, 19
- opening, 20
- opening on project load, 323
- Properties window, 25
- referencing object libraries, 327
- SAVE command, 26
- viewing project data, 20

VBA interface

- advantages
 - with ActiveX, 5
 - with AutoCAD, 3
- elements of, 4
- environment, 3
- implementing in AutoCAD, 4
- VB, similarities with, 3
- with ActiveX interface, 4

VBA Load method, 316

- VBA Manager
 - creating projects, 15
 - defined, 31
 - dialog box, 13
 - embedding projects, 14
 - extracting projects, 15
 - loading projects, 13
 - saving projects, 16
 - unloading projects, 14
- VBA methods, (table), 317
- VBA Object Browser, 327
- VBA programming using ActiveX, 4
- VBA projects
 - acad.dvd* file, 322
 - defined, 12
 - loading on startup, 322
 - running macros on startup, 322
- VBA Show method, 315
- VBA statements
 - On Error, example code, 320
- VBAIDE command, 31
- VBALOAD command, 13, 31
- VBAMAN command, 32
- VBARUN command, 16, 31
- VBASTMT command, 32
- VBAUNLOAD command, 14, 32
- vector files, and raster images, 282
- vectors
 - moving objects, 116
- version, accessing, 36
- Viewport object, 268
 - example code, 252
 - LowerLeftCorner property, 64
 - UpperRightCorner property, 64
- viewports
 - changing views (illustration), 274
 - detail view, 63
 - displaying, 267
 - floating (illustration), 267
 - full view, 63
 - hidden lines, 276
 - horizontal display (illustration), 64
 - in model space, 267
 - in paper space, 268
 - models (illustration), 269
 - modifying, 272
 - properties (list), 271
 - scale factor, 274
 - setting active, 64
 - settings (table), 269
 - splitting, example code, 65
 - tiled (illustration), 267
 - vertical display (illustration), 64
- views
 - creating, 62
- Views collection, 62
- virus protection
 - enabling for macros, 19
 - setting for VBA projects, 19
 - using project code, 14
- Visible property, 58
 - example code, 217
 - setting, example code, 58
- Visual LISP
 - accessing objects, 326
 - calling methods, 326
 - retrieving methods, 326
- Volume property, 255

W

- WBlock method, 290
 - leader lines, 191
- WCS
 - converting coordinates, 249
 - definition, 249
- wedges
 - adding, example code, 255
- Width property, 154, 286
 - example code, 272
- WindowChanged event, 235, 239
- WindowMovedOrResized event, 235, 239
- Windows APIs
 - accessing from VBA, 332
- WindowState property, 58
- WithEvents
 - declaring objects with events, 236, 240
 - enabling objects with events, 243
 - GetObject, VBA function
 - example code, 236
 - VBA Keyword, example code, 236, 240
- WMF files
 - exporting to, 80
- World Coordinate System
 - definition, 249
 - entering coordinates, 246

X

- xdata. *See* extended data
- Xline object, 69
- XLOADCTL system variable, 308
- XRefDemandLoad property, 308
- xrefs. *See* external references

Z

- zero document state, 323
- Zoom Center method
 - illustration, 61

- Zoom window
 - example code, 59
- ZoomAll method, 61
 - example code, 62, 85, 88
 - illustration, 62
- ZoomCenter method, 61
 - example code, 61
- ZoomExtents method
 - example code, 62, 272
 - illustration, 62
 - in active viewport, 61

- zooming
 - defined, 59
 - in paper space (illustration), 275
 - scale factors, 274
 - zoom scales, 275
- ZoomPickWindow method, 59
 - example code, 59
- ZoomScaled method, 60, 274
 - example code, 60, 272
- ZoomWindow method, 59

